МИНОБРНАУКИ РОССИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
Институт дополнительного образования
Высшая инженерная школа

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

КЛАССИФИКАЦИЯ СЕГМЕНТАЦИИ КЛИЕНТОВ

по программе профессиональной переподготовки:
«Анализ данных на языке Python»

Выполнил(а):
Серегин Константин Александрович
Подпись_____

Руководитель:
Кандидат экономических наук,
доцент по научной специальности
«Математические и инструментальные
методы экономики»,
Заграновская Анна Васильевна
Подпись_____

Санкт-Петербург
2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Все люди и похожи, и не похожи друг на друга, поэтому разделение клиентов на группы необходимо для успешных продаж. Не понимая, кто находится перед вами, можно не только потерять клиента, но и создать своей фирме плохую репутацию. При таком раскладе ожидать развития и прибыли очевидно не стоит.

Другое дело, если вы с первых минут можете распознать, кто перед вами, клиент с какими предпочтениями или какой покупательной способностью, от чего она зависит, от возраста, состава семьи или профессии. Видя это, вы с легкостью выстроите тактику общения и определите товар, который будет интересен клиенту, что поспособствует увеличению продаж.

Данная работа продемонстрирует:

- методы обработки имеющейся клиентской базы на языке python;
- использование библиотек Pandas, Numpy и Scikit-learn для поиска закономерностей в данных;
- использование графических библиотек Matplotlib и Seaborn для визуализации выводов;
- построение моделей классификации для определения группы новых клиентов на основе имеющихся признаков;
- элементы кластерного анализа методом k-средних и иерархической кластеризации для поиска наиболее рационального разбиения клиентов на группы;
- автоматизацию поиска лучших признаков, а также необходимого и достаточного их числа с помощью модели случайных лесов, метода SelectKBest(), рекурсивного исключения признаков и метода главных компонент;
- автоматизацию поиска гиперпараметров для моделей линейной, нелинейной классификации и ансамблевых методов с помощью модуля GridSearchCV;

- будут сделаны выводы на основе проделанной работы и даны рекомендации.

1. ИСХОДНЫЕ ДАННЫЕ

Торговая компания планирует выйти на новые рынки со своими существующими продуктами (P1, P2, P3, P4 и P5). После интенсивного исследования рынка они пришли к выводу, что поведение нового рынка похоже на их существующий рынок.

На своем существующем рынке отдел продаж классифицировал всех клиентов на 4 сегмента (A, B, C, D). Затем они выполнили сегментированный охват и коммуникацию для другого сегмента клиентов. Эта стратегия сработала для них исключительно хорошо. Они планируют использовать ту же стратегию для новых рынков.

Данные о клиентах:

№
Пол
Семейное положение
Возраст
Является ли клиент выпускником?
Профессия
Опыт работы в годах
Оценка расходов клиента
Количество членов семьи клиента (включая клиента)
Товар компании
Клиентский сегмент клиента (целевой)

2. ПОСТАНОВКА ЦЕЛИ

Требуется оценить имеющуюся разбивку клиентов на группы, скорректировать сегментацию рынка если требуется и обучить модель для классификации новых клиентов.

3. ПРЕДВАРИТЕЛЬНЫЙ АНАЛИЗ ДАННЫХ

3.1 Типы данных.

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1	Segmentation
0	Male	No	22	No	Healthcare	1.0	Low	4.0	Cat_4	D
1	Female	Yes	38	Yes	Engineer	NaN	Average	3.0	Cat_4	A
2	Female	Yes	67	Yes	Engineer	1.0	Low	1.0	Cat_6	B
3	Male	Yes	67	Yes	Lawyer	0.0	High	2.0	Cat_6	B
4	Female	Yes	40	Yes	Entertainment	NaN	High	6.0	Cat_6	A
...	...	...	...	...	...	...	...	...	...	...
8063	Male	No	22	No	NaN	0.0	Low	7.0	Cat_1	D
8064	Male	No	35	No	Executive	3.0	Low	4.0	Cat_4	D
8065	Female	No	33	Yes	Healthcare	1.0	Low	1.0	Cat_6	D
8066	Female	No	27	Yes	Healthcare	1.0	Low	4.0	Cat_6	B
8067	Male	Yes	37	Yes	Executive	0.0	Average	3.0	Cat_4	B

8068 rows × 10 columns

Рисунок 3.1 – Исходные данные

Исходные данные представляют собой таблицу, состоящую из 8068 клиентов, классифицированных по 9 признакам, большинство из которых имеют тип object, но по своей сути имеют категориальный тип. Для дальнейшей работы потребовалось произвести перекодировку данных в числовые значения, включая результативный признак, что было проделано пошагово с помощью метода np.where (см. приложение п.1), в результате все данные получили вид:

	Пол	Семейное_положение	Возраст	Высшее_образование	Профессия	Опыт_работы	Уровень_расходов	Размер_семьи	Анонимная_категория	Целевая_категория
0	1	0.0	22	0.0	2.0	1.0	3	4.0	4.0	4
1	0	1.0	38	1.0	4.0	NaN	2	3.0	4.0	1
2	0	1.0	67	1.0	4.0	1.0	3	1.0	6.0	2
3	1	1.0	67	1.0	6.0	0.0	1	2.0	6.0	2
4	0	1.0	40	1.0	3.0	NaN	1	6.0	6.0	1
...	...	...	...	...	...	...	...	...	...	...
8063	1	0.0	22	0.0	NaN	0.0	3	7.0	1.0	4
8064	1	0.0	35	0.0	7.0	3.0	3	4.0	4.0	4
8065	0	0.0	33	1.0	2.0	1.0	3	1.0	6.0	4
8066	0	0.0	27	1.0	2.0	1.0	3	4.0	6.0	2
8067	1	1.0	37	1.0	7.0	0.0	2	3.0	4.0	2

8068 rows × 10 columns

Рисунок 3.2 – Данные после обработки методом np.where

3.2 Результативный признак.

Для построения ряда моделей важна равномерность распределения результативного признака, в данном наборе результативные признаки распределены равномерно, что видно на диаграмме (см. приложение п.3):

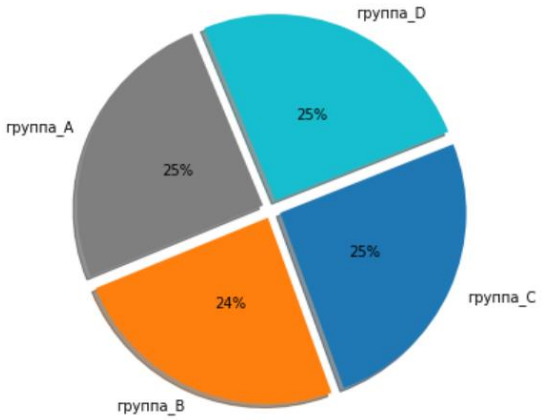


Рисунок 3.3 – Распределение результативных признаков

3.3 Наличие выбросов.

Выбросы отсутствуют.

3.4 Наличие пропусков и работа с ними.

Как можно видеть из диаграммы пропуски присутствуют в большом количестве в половине столбцов.

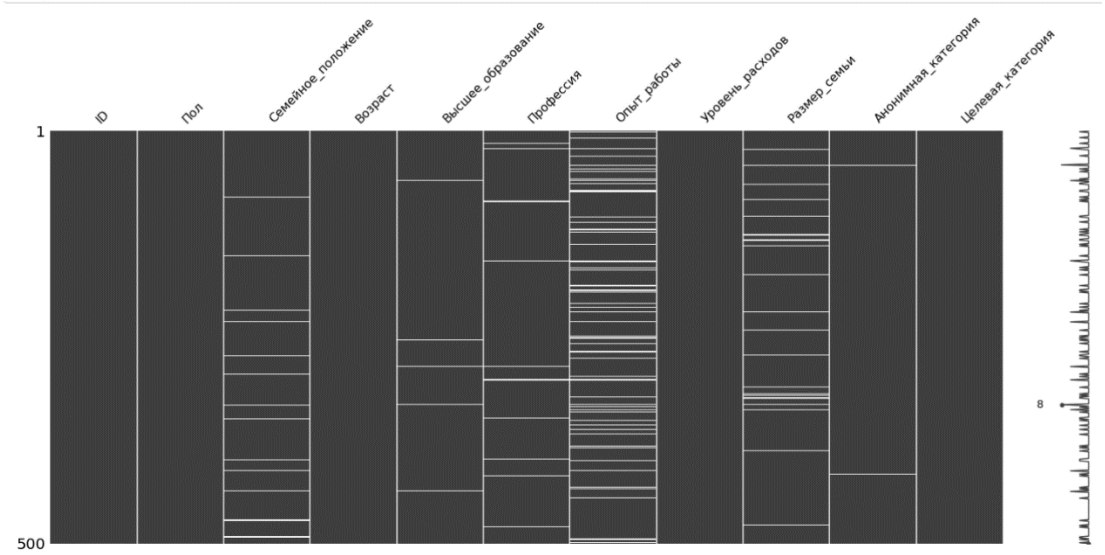


Рисунок 3.4 – Пропуски в данных

Для обработки пропусков были выбраны 4 стратегии:

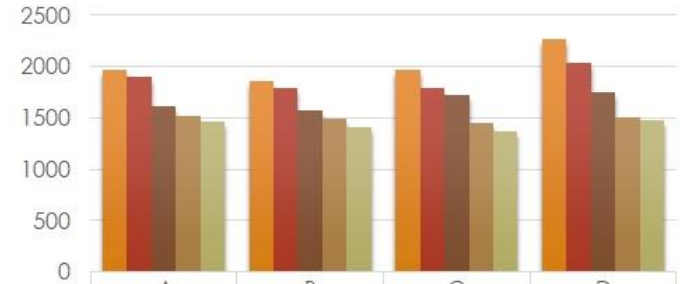
1. Заполнение пропущенных данных (см. приложение п.2) путем использования функции fillna, после группировки по результативному признаку и ряду факторных признаков с последующим удалением дубликатов (метод drop_dublicates()).

2. Удаление строк с пропущенными данными (функция dropna).

3. Выполнение п.2 и последующее удаление дубликатов (метод drop_dublicates()).

4. Выполнение п.2, п.3 и последующее удаление противоречивых данных (данных имеющие одинаковые факторные признаки и разные результативные).

Все 4 способа показали, что объем данных для анализа и построения моделей остается достаточным, а распределение категорий равномерным, что наблюдается на графике, где можно увидеть 4 результативных признака и уменьшение их количества в зависимости от выбранного метода обработки (см. приложение п.5).



	A	B	C	D
Ряд 1	1972	1858	1970	2268
Ряд 2	1902	1794	1793	2039
Ряд 3	1616	1572	1720	1757
Ряд 4	1519	1489	1454	1505
Ряд 5	1468	1407	1375	1477

Рисунок 3.5 – Распределение результативных признаков A, B, C, D в зависимости от метода удаления пропусков.

Ряд 1 - Исходные данные без удаления пропусков

Ряд 2 – Удаление пропусков первым методом (метод заполнения)

Ряд 3 - Удаление пропусков вторым методом

Ряд 4 - Удаление пропусков третьим методом

Ряд 5 - Удаление пропусков четвертым методом

3.5 Выбор стратегии удаления пропусков.

По результатам обработки пропусков получилось 4 группы данных, из которых нужно выбрать лучшую. Для этого решено было предварительно обучить ряд моделей с параметрами по умолчанию, выбрать лучшую и на основе ее сделать прогноз доли правильных ответов для всех 4-х групп.

В результате предварительного опробования моделей очень хорошо себя зарекомендовала модель RandomForestClassifier на основе нее и будет отобран лучший набор признаков (см. приложение п.6).

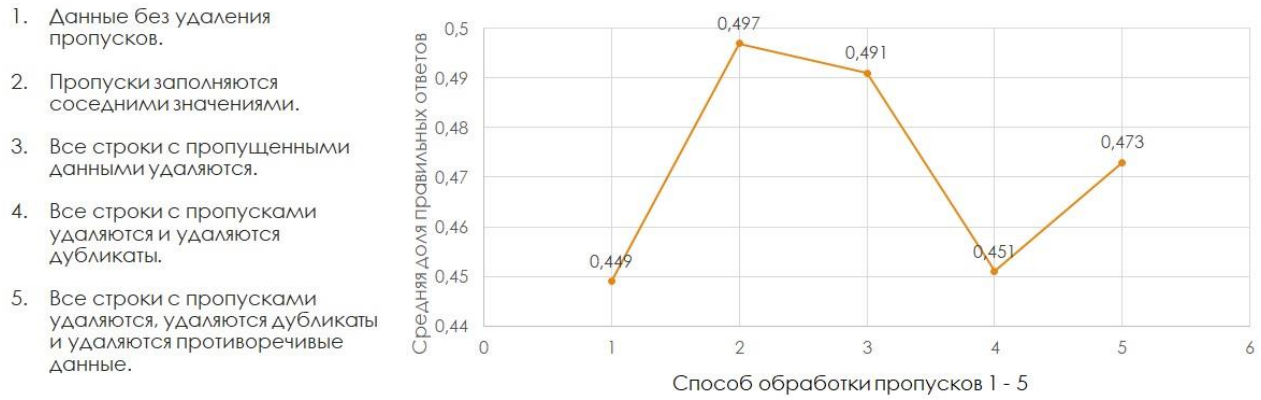


Рисунок 3.6 – Доля правильно предсказанных ответов моделью случайных лесов в зависимости от набора данных.

По диаграмме Рис.3.6 видно, что доля правильных ответов при использовании набора 2, где данные заполняются с помощью функции fillna() и удаляются дубликаты - выше по сравнению с остальными данными, исходя из этого в последующих вычислениях будет использоваться именно этот набор.

3.6 Отбор главных признаков и их количества.

Обор числа признаков также производился на основе модели RandomForestClassifier с использованием метода SelectKBest(), рекурсивного исключения признаков и метода главных компонент.

```
model=RandomForestClassifier()
for i in lst_2:
   kfold=KFold(n_splits=i,random_state=7,shuffle=True)
   for j in lst_1:
      test=SelectKBest(score_func=f_classif, k=j)
      fit=test.fit(X,Y)
      features=fit.transform(X)
      results=cross_val_score(model, features, Y, cv=kfold)
      res = results.mean()
      k.append(res)
      print(f'n_splits: {i} KBest: {j} {res}')
   print(f'i {i} max: {max(k)}')
```

```
n_splits: 7 KBest: 1 0.4006059401725013
n_splits: 7 KBest: 2 0.4409634262939173
n_splits: 7 KBest: 3 0.4532667736559472
n_splits: 7 KBest: 4 0.4670693041119911
n_splits: 7 KBest: 5 0.4418598499210807
n_splits: 7 KBest: 6 0.4798209420179405
n_splits: 7 KBest: 7 0.4729208577449868
n_splits: 7 KBest: 8 0.4816186705532161
n_splits: 7 KBest: 9 0.49422339764867124
i 7 max: 0.49422339764867124
```

Рисунок 3.7 – Пример кода для отбора признаков методом SelectKBest() на основе модели RandomForestClassifier.

На Рис.3.7 показано как велся отбор признаков и создавался список с данными для построения диаграмм на Рис.3.8 (все примеры кода см. приложение п.7, 8, 9).

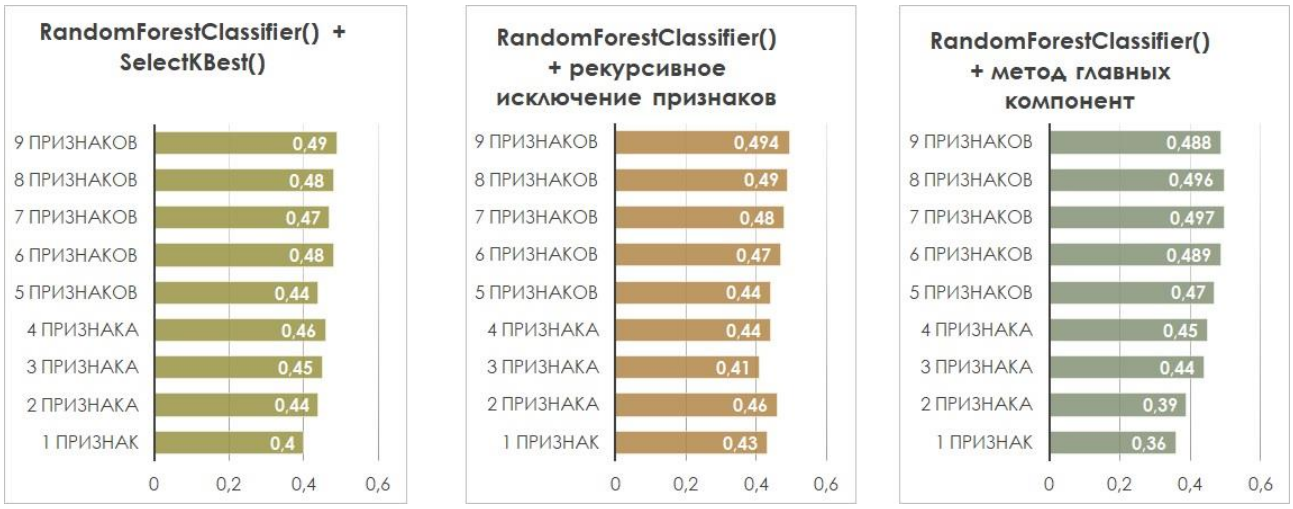


Рисунок 3.8 – Отбор признаков.

Из диаграмм на Рис.3.8 видно, что модель предсказывает лучше всего при наличии от 6 до 9-ти лучших признаков, а при меньшем количестве признаков качество значительно падает.

Самыми существенными признаками оказались (по уменьшению значимости) «Возраст», «Семейное положение», «Образование» и «Размер_семьи». Самым бесполезным - «Пол», «Опыт работы» и «Категория товара».

На основе полученных данных можно заключить, что при необходимости 3 признака можно отбросить без потери качества модели, но в данной работе мы будем использовать все 9 признаков исходя из того, что они были включены в набор данных и не были отброшены ранее.

3.7 Корреляционная матрица.

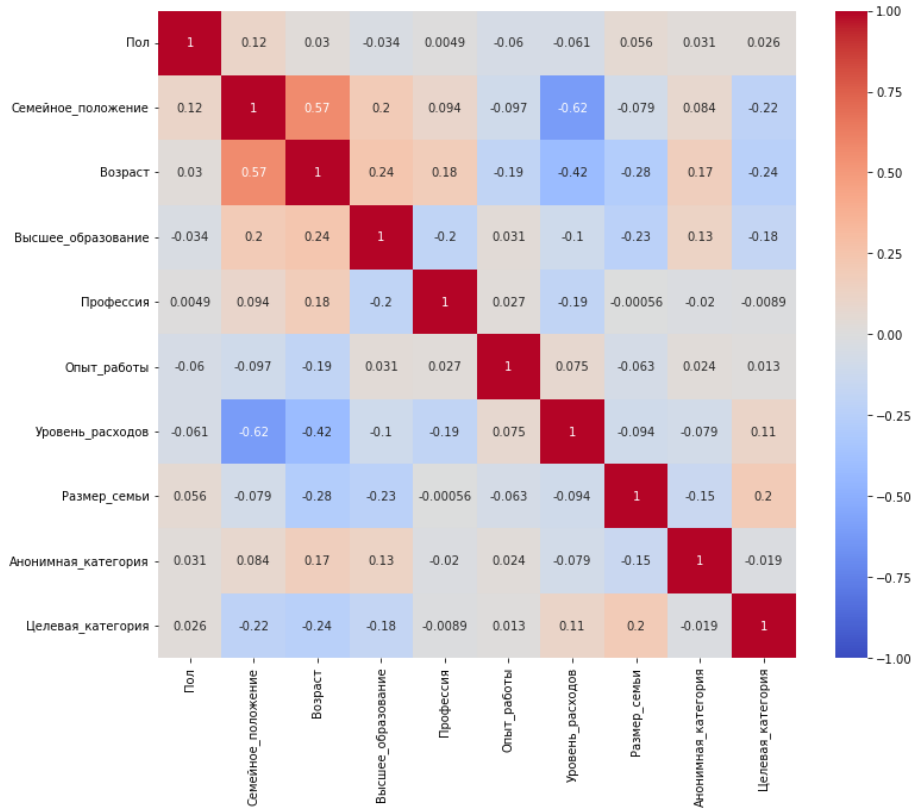


Рисунок 3.9 – Корреляционная матрица.

На Рис.3.9 видно, что корреляция между результативным и факторными признаками очень низкая (не превышает 0.24), что косвенно может указывать на то, что качество предсказания моделей будет не высоко.

Корреляция между самими факторными признакам тоже не высока, из чего можно заключить, что использование факторного анализа для понижения размерности не целесообразно.

3.8 Резюме.

В результате предварительного анализа и обработки данных были перекодированы признаки из строчных в числовые значения, удалены пропуски, выполнено предварительное обучение модели случайных лесов для первичного знакомства с данными и способностью их к обучению. Выполнен отбор признаков, выявлены наиболее важные признаки и незначительные.

На основе корреляционной матрицы была обнаружена низкая корреляция между факторными признаками, что указало на то, что применение факторного анализа для снижения размерности в данном случае не целесообразно. Также была выявлена низкая корреляция между факторными признаками и результативным, что может в первом приближении говорить о том, что качество моделей классификации будет недостаточно высоко.

4. КЛАСТЕРНЫЙ АНАЛИЗ

Целью кластерного анализа на данном этапе является определение связи результативных и факторных признаков. Предполагается, что в случае хорошей связи, кластеры, определенные методом k-средних, и результативные признаки будут примерно совпадать, в идеале – будет наблюдаться полное совпадение. В обратном случае будет сделан вывод о том, что факторные и результативные признаки связаны слабо.

Если на основе корреляционной матрицы можно сделать вывод о слабой связи признаков по-отдельности, то здесь можно сделать вывод о слабой связи всех в совокупности факторных признаков и результативного.

Путем иерархической кластеризации будет проверена целесообразность разделения клиентов на 4 группы.

4.1 Кластерный анализ с помощью метода k-средних.

С помощью метода KMeans была выполнена разбивка данных на 4 кластера на основе всех признаков (см. приложение п.10), в итоге получили целочисленный ряд значений от 1 до 4.

Для определения совпадения кластеров и результативных признаков полученный ряд был добавлен в исходный датасет, проведена группировка данных по результативному признаку, с помощью value_counts найдена доля кластеров для каждого результативного признака и с помощью matplotlib построены 4 круговых диаграммы.

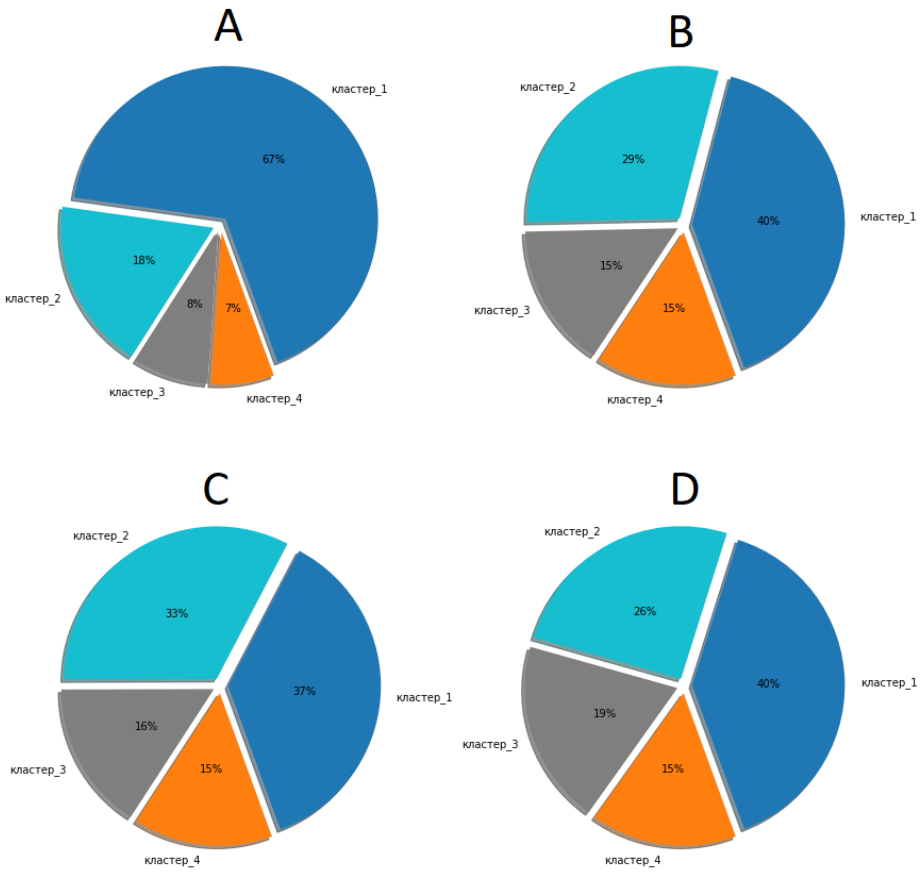


Рисунок 3.10 – Диаграммы пересечения результативных признаков и кластеров.

Опираясь на диаграммы Рис.3.10 можно сделать вывод о том, что кластеры и результативные признаки не совпадают, причем абсолютно, в случае полного совпадения данные диаграммы имели бы разные цвета (к примеру, первая – синяя, вторая – бирюзовая, третья – серая, четвертая - оранжевая).

Дополнительно была проведена проверка совпадения результативных признаков и кластеров путем изменения числа факторных признаков совмещением метода SelectKBest и метода KMeans (см. приложение п.11), в результате было выяснено, что вне зависимости от числа признаков совпадений не наблюдается и все диаграммы схожи (Рис. 3.11), разночтения колеблются в пределах ±10%.

Рисунок 3.11 – Результаты кластерного анализа в зависимости от числа признаков (совместное применение SelectKBest и KMeans).

В результате можно сделать вывод о том, что факторные признаки и результативный имеют слабую связь, кластерный анализ показал, что существующая взаимосвязь признаков подсказывает разбиение несколько иным образом.

4.2 Иерархический кластерный анализ.

Проводился с помощью функции linkage (см. приложение п.12).

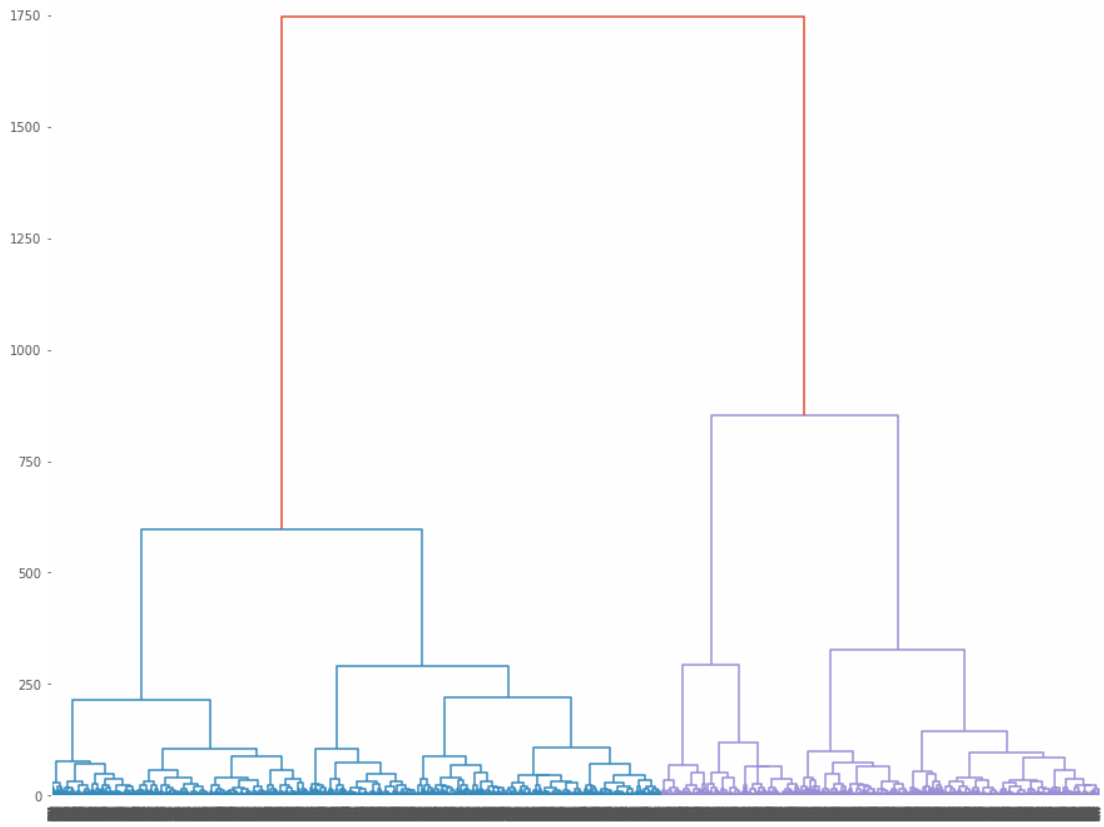


Рисунок 3.12 – Дендрограмма

Полученная дендрограмма говорит о том, что на основе имеющихся признаков данные целесообразно разбить на 2, 3, 4 и т.д. кластера, а значит разбивка клиентов на 4 группы вполне возможна и не противоречит полученным результатам.

4.2 Резюме.

Таким образом по результатам исследования можно сделать 2 вывода: о том, что разбивка клиентов на группы не совсем рациональна независимо от числа признаков и о том, что разбивка на 4 группы уместна и не противоречит полученным данным.

5. ПОСТРОЕНИЕ МОДЕЛЕЙ

Обучение моделей классификации для распределения по группам новых клиентов является одной из главных задач.

Для обработки данных будут использованы:

Линейные модели:

- модель логистической регрессии LogisticRegression
- метод опорных векторов SVC

Нелинейные модели:

- метод kNN KNeighborsClassifier
- модель решающего дерева DecisionTreeClassifier

Ансамблевые методы:

- метод пакетирования BaggingClassifier
- метод случайного леса RandomForestClassifier
- метод дополнительных деревьев ExtraTreesClassifier
- метод AdaBoostClassifier
- метод GradientBoostingClassifier
- метод XGBClassifier

Последовательность действий:

1. Обучение моделей с гиперпараметрами заданными по умолчанию и перебором признаков от 1 до 9 с помощью SelectKBest, построение графиков с помощью matplotlib на основе полученных данных.
2. Обучение моделей с гиперпараметрами подобранными с помощью GridSearchCV и перебором признаков от 1 до 9 с помощью SelectKBest, построение графиков с помощью matplotlib на основе полученных данных.
3. Оценка полученных результатов.

5.1 Обучение моделей с базовыми параметрами.

Целью было не только обучить модели, но и максимально автоматизировать процесс, для этого был создан список с моделями и пустой список k, далее путем перебора моделей и признаков от 1 до 9 список k был заполнен значениями «средней доли правильных ответов» для каждой модели для различного числа признаков (см. Рис.3.13) и затем путем индексации с помощью matplotlib были построены графики (полный код см. приложение п.14).

В качестве метрик качества были использованы функции classification_report, confusion_matrix(), model.score() и model.mean()

```
lst_1 = list(range(1,10))
lst_3 = [LogisticRegression(max_iter = 5000), DecisionTreeClassifier(), AdaBoostClassifier(),SVC(),
         KNeighborsClassifier(),RandomForestClassifier(),ExtraTreesClassifier(),
         XGBClassifier(seed = 7, n_estimators = 100, max_depth = 6, learning_rate = 0.3),
        GradientBoostingClassifier(n_estimators=20, random_state=7),
        BaggingClassifier()]
k = []
```

```
for i in lst_3:
   model = i
   for j in lst_1:
        test=SelectKBest(score_func=f_classif, k=j)
        fit=test.fit(X,Y)
        features=fit.transform(X)
        X_train, X_test, Y_train, Y_test = train_test_split(features, Y, test_size=0.25,
                                                            shuffle=True, random_state=7)
        model.fit(X_train,Y_train)
        result=model.score(X_test,Y_test)
        k.append(result)
        print(result)
        print(f'Модель: {i} Количество  признаков = {j}')
```

```
Модель: KNeighborsClassifier() Количество  признаков = 1
0.385122975404919
Модель: KNeighborsClassifier() Количество  признаков = 2
0.4295140971805639
Модель: KNeighborsClassifier() Количество  признаков = 3
0.4451109778044391
```

Рисунок 3.13 – Создание списка со значениями model.score() для всех моделей с различным числом признаков.

```python
import matplotlib.pyplot as plt
import seaborn as sns
x = 1,2,3,4,5,6,7,8,9
plt.figure(figsize=(16, 16))

line1, = plt.plot(x, k[:9], 'o-b',label='Логистическая регрессия')
line2, = plt.plot(x, k[9:18], 'o-.m', label='Модель решающего дерева')
line3, = plt.plot(x, k[18:27], 'o--g', label='AdaBoost(адаптивное усиление)')

line4, = plt.plot(x, k[27:36], 'o:r', label='Метод опорных векторов')
line5, = plt.plot(x, k[36:45], 'o-c', label='Метод KNN')
line6, = plt.plot(x, k[45:54], 'o-.y', label='Случайный лес')
line7, = plt.plot(x, k[54:63], 'o--k', label='Дополнительных деревьев')
line8, = plt.plot(x, k[63:72], 'o:',color='orange', label='XGBClassifier')
line9, = plt.plot(x, k[72:81], 'o:',color='brown', label='GradientBoostingClassifier')
line10, = plt.plot(x, k[81:], 'o:',color='pink', label='BaggingClassifier')

plt.text(0.5, 0.5, 'text')
plt.grid()
plt.legend(fontsize=10)
plt.show()
```

Рисунок 3.14 – Создание графиков на основе списка со значениями model.score().

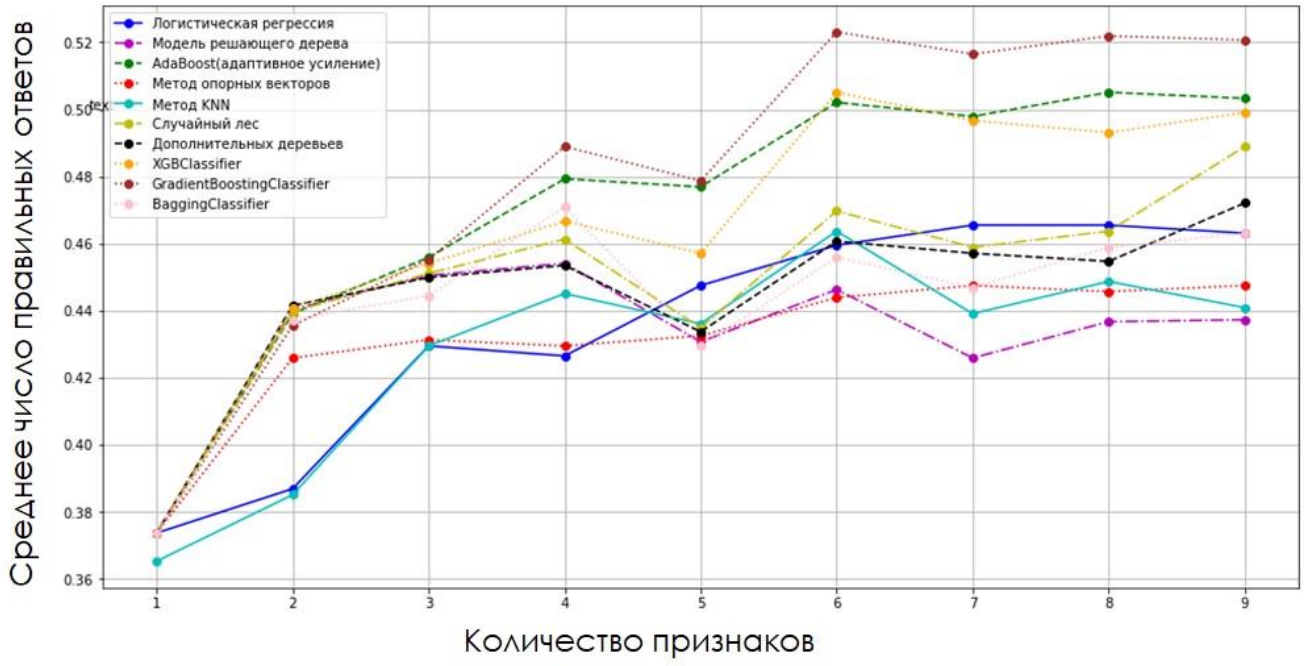


Рисунок 3.15 – Зависимость качества модели с базовыми гиперпараметрами от числа признаков

По графику Рис.3.15 можно заметить, что с увеличением числа признаков растет и качество моделей, скачек наблюдается на уровне 6-ти признаков, как и

было предсказано на стадии предварительного анализа. Наилучшее качество показывают ансамблевые методы, но среднее число правильных ответов не превышает 53%.

5.2 Поиск наилучших гиперпараметров с помощью GridSearchCV и обучение моделей.

Для поиска гиперпараметров производилась (см. приложение п.13):

1. Разбивка данных на тренировочную и тестовую выборки;
2. Создавался список с предлагаемыми гиперпараметрами;
3. С помощью GridSearchCV производился поиск на основе заданной модели;
4. С помощью grid.best_estimator_ выводился результат поиска;
5. Производилось предсказание на тестовой выборке с помощью grid.predict();
6. Производилась оценка качества модели с помощью функции classification_report(), confusion_matrix() и model.mean();
7. Модель с новыми гиперпараметрами добавлялась в список для последующей обработки и построения графиков.

```
lst_1 = list(range(1,10))
lst_3 = [LogisticRegression(C=0.01, max_iter=5000),
         DecisionTreeClassifier(criterion='entropy', max_depth=5),
         AdaBoostClassifier(learning_rate=0.11, n_estimators=60),
         SVC(C=100, gamma=0.001),
         KNeighborsClassifier(metric='manhattan', n_neighbors=50),
         RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60),
         ExtraTreesClassifier(max_depth=37,
                              min_weight_fraction_leaf=0.001,
                              n_estimators=52),
         XGBClassifier(learning_rate=0.2, max_depth=5, max_features='log2',
                       min_samples_leaf=0.1, min_samples_split=0.1,
                       n_estimators=10),
         GradientBoostingClassifier(n_estimators=100, random_state=7),
         BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entropy',
                                                                 max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100)]
k = []
```

Рисунок 3.16 – Список моделей обученных GridSearchCV

В результате обработки моделей был построен график по аналогии с Рис. 3.15.

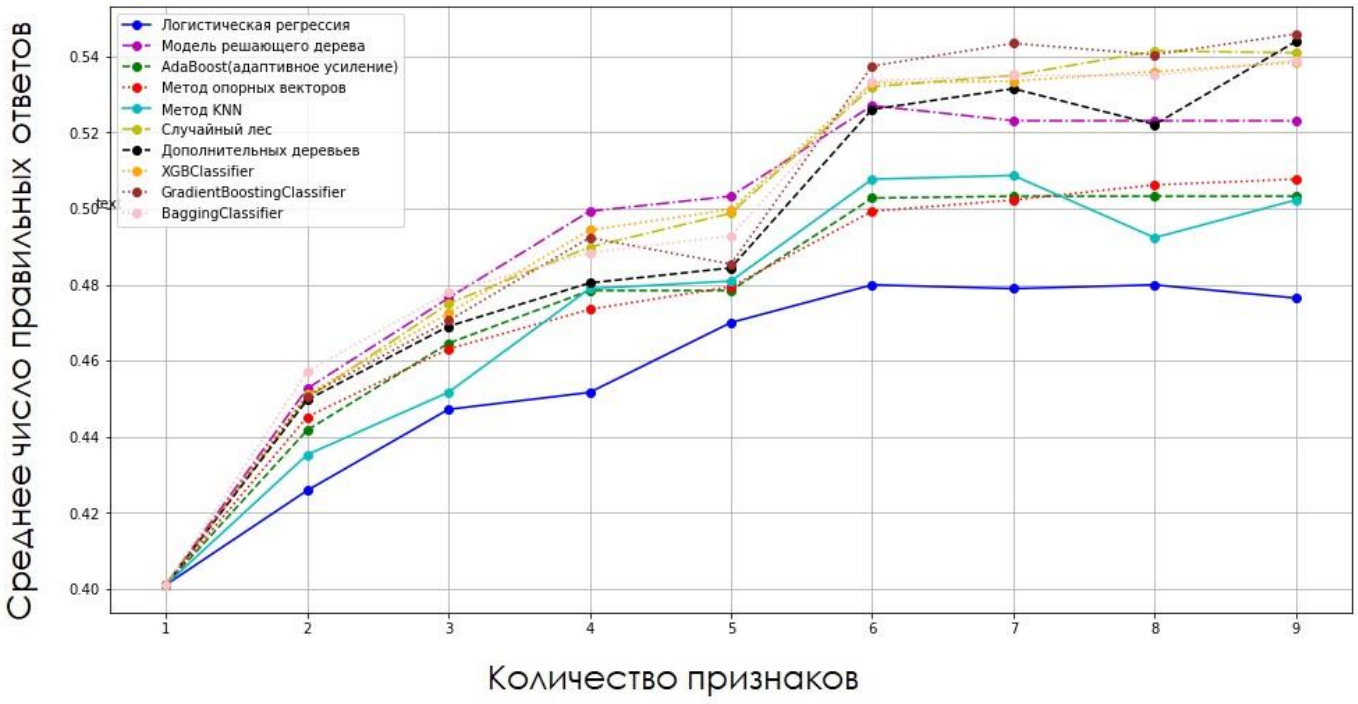


Рисунок 3.17 – Зависимость качества модели с отобранными GridSearchCV параметрами от числа признаков.

5.3 Резюме.

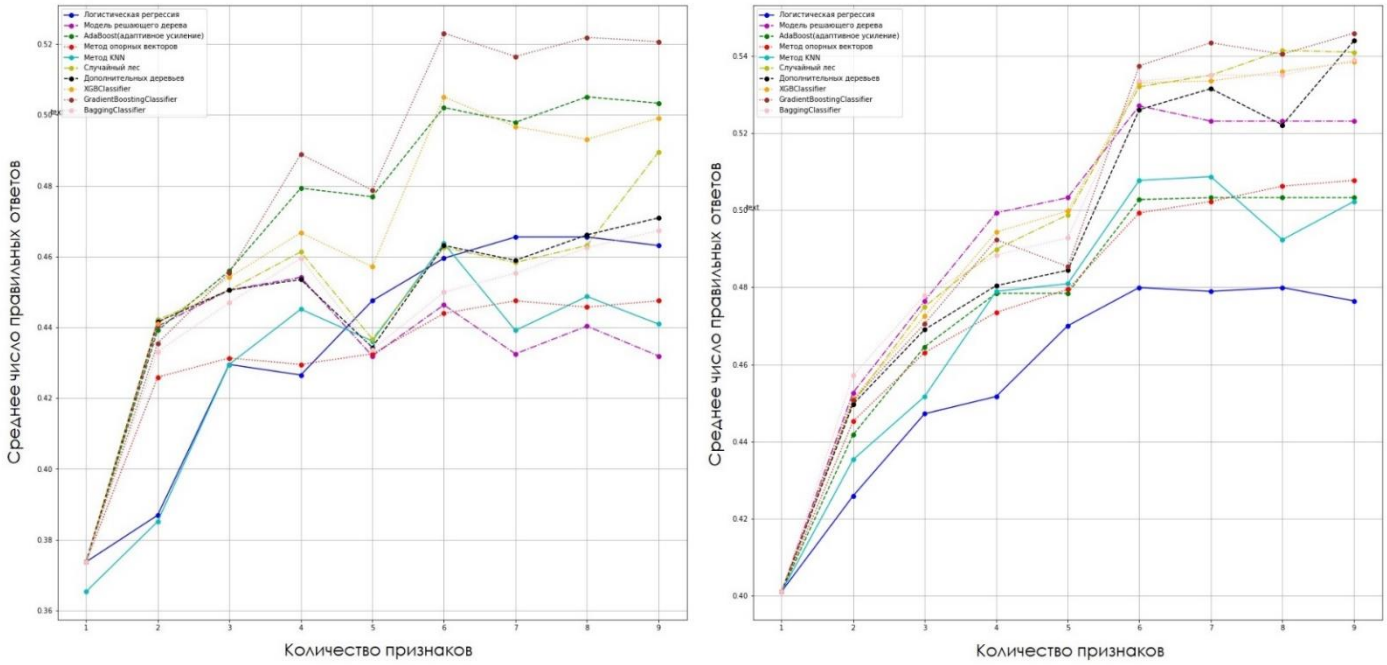


Рисунок 3.18 – Качество моделей до использования GridSearchCV (слева) и после (справа).

При сравнении 2-х групп графиков можно заметить большую выраженность увеличения качества моделей при достижении 6-ти признаков в модели справа. Число правильно предсказаний увеличилось с 52% (1 модель) до 54 – 55% (4 модели).

Лучше всего себя зарекомендовал метод GradientBoostingClassifier().

На данном этапе можно сделать заключение о том, что модели получаются недостаточно качественными, 55% правильных ответов это очень мало, пользы от таких предсказаний не много и находить какие-либо закономерности с целью выбора стратегии для продвижения продукции бессмысленно. Можно сделать предположение, что какие-то важные признаки, которые повлияли на классификацию были исключены из таблицы.

С целью создания более качественной модели на основе имеющихся признаков предлагается вернуться к кластерному анализу.

6. КЛАССИФИКАЦИЯ КЛИЕНТОВ НА ОСНОВЕ КЛАСТЕРНОГО АНАЛИЗА

Напомню, что кластеризация — это группировка множества объектов на подмножества (кластеры) таким образом, чтобы объекты из одного кластера были более похожи друг на друга, чем на объекты из других кластеров по какому-либо критерию.

Кластеризация проводилась путем разбивки на 4 группы (см. приложение п.15), правомерность данной разбивки подтверждена иерархическим кластерным анализом (см. Рис.3.12), методом k-средних на основе 9-ти признаков.

На основе полученных результатов была обучена модель GradientBoostingClassifier().

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	572
1.0	1.00	1.00	1.00	255
2.0	1.00	1.00	1.00	528
3.0	1.00	0.99	1.00	662
accuracy			1.00	2017
macro avg	1.00	1.00	1.00	2017
weighted avg	1.00	1.00	1.00	2017

Рисунок 3.19 – определение качества модели с помощью функции classification_report().

```
[[571   0   0   1]
 [  0 255   0   0]
 [  0   0 528   0]
 [  5   0   0 657]]
```

Рисунок 3.20 – определение качества модели с помощью функции confusion_matrix().

Оценка модели с помощь classification_report (Рис.3.19) и confusion_matrix (Рис.3.20) показала, что модель определяет принадлежность клиента практически со 100% вероятностью.

cluster	Пол	Семейное_положение	Возраст	Высшее_образование	Профессия	Опыт_работы	Уровень_расходов	Размер_семьи	Анонимная_категория	Целевая_категория
1	0.519298	0.616541	39.005514	0.768421	3.046115	3.528822	2.472682	2.495238	5.095739	2.196992
2	0.534489	0.181114	26.625231	0.376614	3.320177	3.030247	2.840649	3.462929	4.897824	3.081151
3	0.581767	0.951128	74.921992	0.625000	4.957707	1.118421	1.946429	2.038534	5.599624	2.328008
4	0.571180	0.859817	52.659556	0.784502	2.962995	2.075316	2.215934	2.794515	5.252068	2.375272

Рисунок 3.21 – Средние значения признаков для каждого кластера.

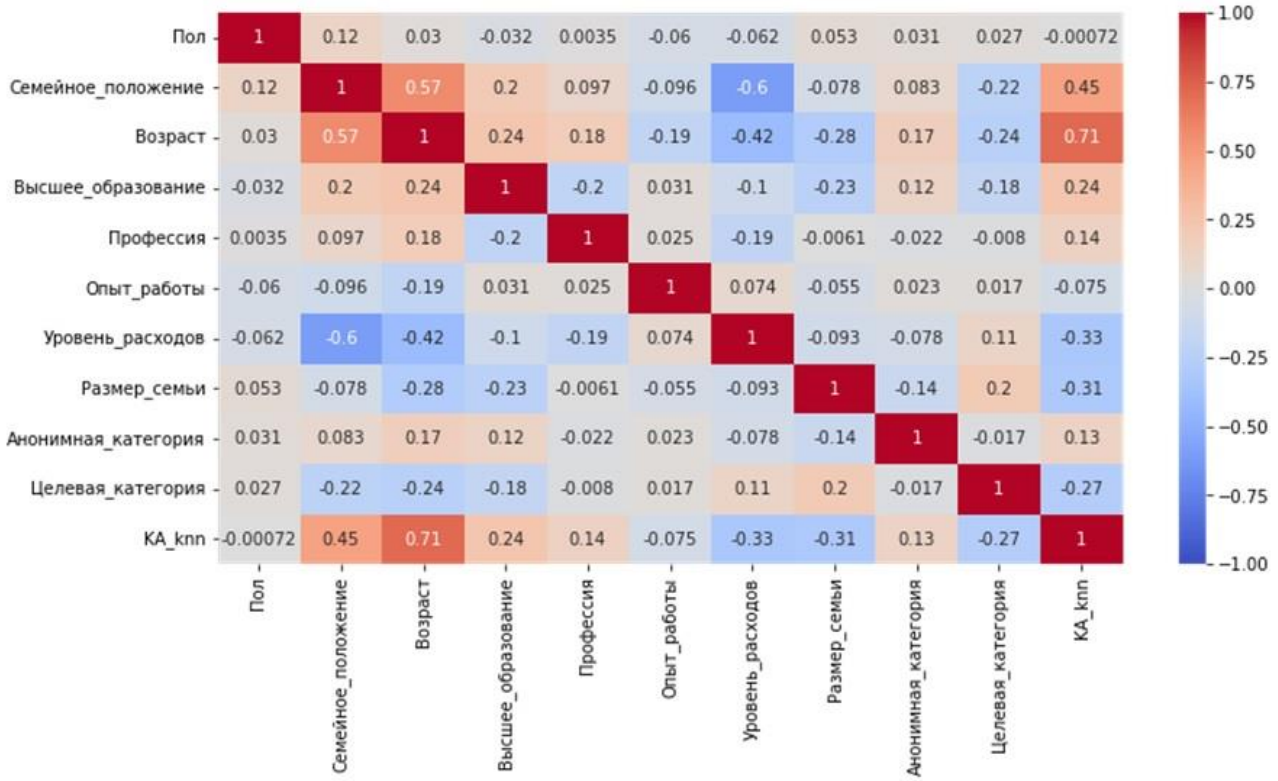


Рисунок 3.22 – Корреляционная матрица.

При рассмотрении корреляционной матрицы Рис.3.22 можно отметить, что наибольшее значение имеет «Возраст» клиента (0.71), а из Рис.3.21 видно, что возраст четко разделяет клиентов по кластерам, на основе этих данных можно обозначить группы на основе возрастного критерия:

Кластер 1 – Зрелый возраст (≈ 39 лет)

Кластер 2 – Молодежь (≈ 26 лет)

Кластер 3 – Пожилые люди (≈ 75 лет)

Кластер 4 – Средний возраст (≈ 52 года)

Корреляционная матрица показывает, что для кластеров и «Целевой_категории» сохраняется последовательность важности признаков, а это является важным результатом кластерного анализа, т.к. демонстрирует, что новый и старый способ разделения клиентов основывались на одних и тех же принципах, что позволяет сохранить прежние закономерности.

ЗАКЛЮЧЕНИЕ

В заключении хотелось бы рекомендовать в качестве классификации новых клиентов использовать результаты кластерного анализа, т.к. повышаются корреляции между результативным и факторными признаками, оставляя корреляции между факторными признаками на том же уровне, что позволяет обучать более качественные модели классификации, а это приводит к тому, что клиент будет с высокой вероятностью правильно причислены к той или иной группе (в нашем случае 100%-й результат).

Т.к. мы обеспечим правильную классификацию клиентов, то данные можно группировать по различным признакам, находить закономерности, делать выводы и, например, рекомендовать тот или иной товар, не опасаясь, что новый клиент попал не в ту группу.

СПИСОК ЛИТЕРАТУРЫ

1. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.
2. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.
3. Наследов, А.Д. Математические методы психологического исследования. Анализ и интерпретация данных. Учебное пособие [Текст]. – СПб: Речь, 2004. – 392 с.
4. Дуброва Т.А. Статистические методы прогнозирования: Учебное пособие для вузов. М. ЮНИТИ-ДАНА, 2003
5. Айвазян, С.А. Прикладная статистика: Классификация и снижение размерности: Справ.изд. [Текст]/ С.А.Айвазян, В.М.Бухштабер, И.С.Енюков, Л.Д.Мешалкин; под ред. С.А.Айвазяна. – М.: Финансы и статистика, 1989. – 607 с.
6. Гласс Дж. Статистические методы в педагогике и психологии [Текст] / Гласс Дж., Стэнли Дж. – М.: Прогресс, 1976. – 496 с.
7. Абдрахманов М.И., Python. Визуализация данных. Matplotlib. Seaborn. Mayavi. 2020. - 412 с.
8. Крис Элбон, Машинное обучение с использованием Python. Сборник рецептов. 2018. - 378 с.
9. Артем Груздев, Майкл Хейдт, изучаем pandas. Высокопроизводительная обработка и анализ данных в Python. 2017. - 684 с.
10. П. Брюс, Э. Брюс, Практическая статистика для специалистов Data Science: Пер. с англ. / — СПб.: БХВ-Петербург, 2018. — 304 с.

ПРИЛОЖЕНИЕ

1. Пример кодирования столбцов адресно (т.е. каждому значению в столбце присваиваем конкретное число)

Профессия

```
df['Профессия']= np.where(df['Профессия']=='Artist',1,
          (np.where(df['Профессия']=='Healthcare',2,
          (np.where(df['Профессия']=='Entertainment',3,
          (np.where(df['Профессия']=='Engineer',4,
          (np.where(df['Профессия']=='Doctor',5,
          (np.where(df['Профессия']=='Lawyer',6,
          (np.where(df['Профессия']=='Executive',7,
          (np.where(df['Профессия']=='Marketing',8,
          (np.where(df['Профессия']=='Homemaker',9,np.nan))))))))))))))))
)
```

2. Заполняем NaN соседними значениями с предварительной группировкой

```
df_fillna = df

df_fillna = df_fillna.sort_values(by=['Целевая_категория','Возраст',])

df_fillna['Семейное_положение'].fillna(method='ffill', inplace=True)

df_fillna = df_fillna.sort_values(by=['Целевая_категория','Возраст',
                                      'Семейное_положение'])

df_fillna['Высшее_образование'].fillna(method='ffill', inplace=True)

df_fillna['Профессия'].fillna(method='ffill', inplace=True)

df_fillna['Размер_семьи'].fillna(method='ffill', inplace=True)

df_fillna['Анонимная_категория'].fillna(method='ffill', inplace=True)

df_fillna['Опыт_работы'].fillna(method='ffill', inplace=True)

df_fillna = df_fillna.sort_values(by=['Целевая_категория','Возраст'])
```

3. Изображение распределения результативных признаков на основе matplotlib

```
        # Инициализация данных
data = df_T.Целевая_категория.value_counts()
labels = ['группа_C', 'группа_D', 'группа_A', 'группа_B']

# Настройка цветовой гаммы
```

```
colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

# Рендеринг круговой диаграммы
g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
        labels=labels, pctdistance=0.5, shadow =True, startangle = 290
       )

# Название рисунка
plt.figtext(0.05, -0.1, 'Рисунок 1 – Распределение групп в результативном призн
ake \n                    после удаления пропусков', \
            fontsize=16, color='black',fontweight='bold'
            )
plt.show()
```

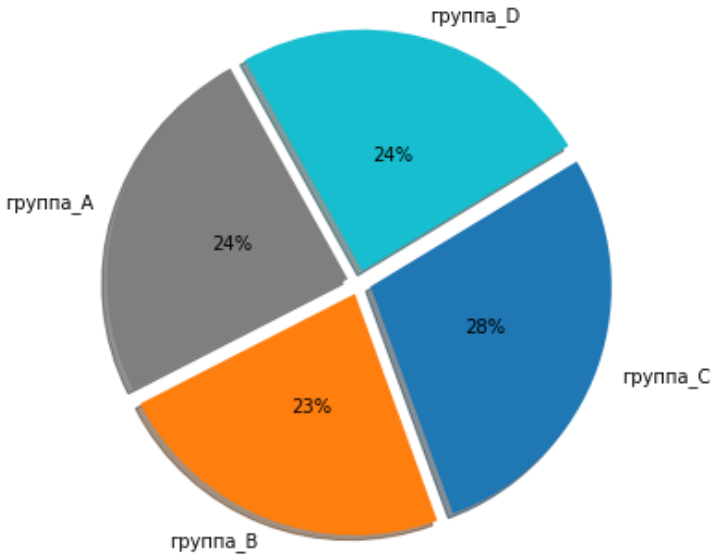
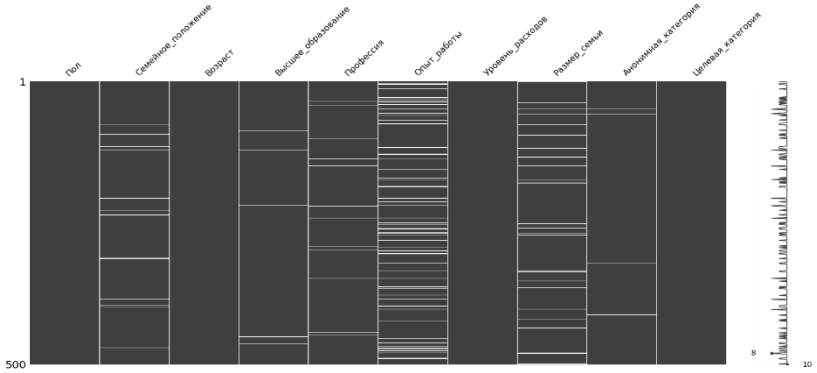


Рисунок 1 — Распределение групп в результативном признаке после удаления пропусков

4. Изображение пропусков с помощью библиотеки missingno

```
import missingno as msno
ax= msno.matrix(df.sample(500))
```



5. Визуализация изменения числа результативных признаков в зависимости от выбранного метода очистки

```
df_1.Целевая_категория.value_counts()
```

```
       4    2039
1    1902
2    1794
3    1793
```

```
df_2.Целевая_категория.value_counts()
```

```
       4    2268
1    1971
3    1970
2    1858
```

```
df_3.Целевая_категория.value_counts()
```

```
       4    1757
3    1720
1    1616
2    1572
```

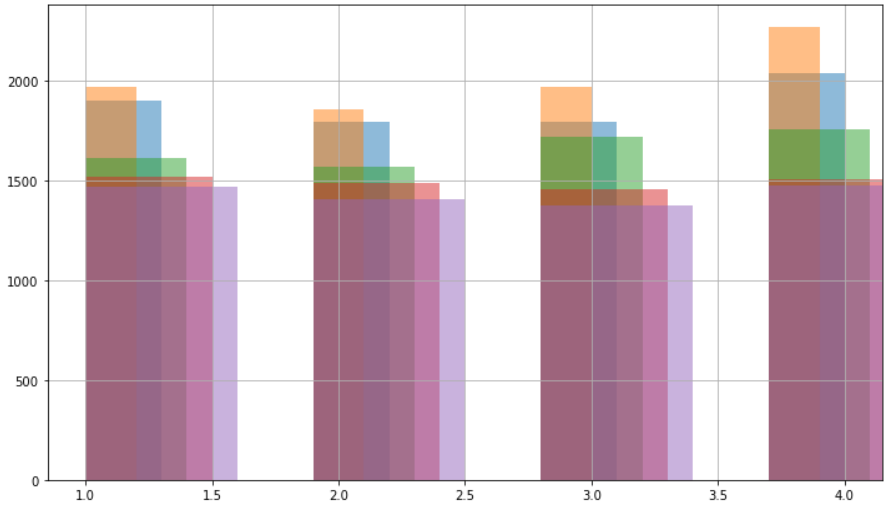
```
df_4.Целевая_категория.value_counts()
```

```
       1    1519
4    1505
2    1489
3    1454
```

```
df_5.Целевая_категория.value_counts()
```

```
       4    1477
1    1468
2    1407
3    1375
```

```
        plt.figure(figsize=(12, 7))
df_1.Целевая_категория.hist(width = 0.3,alpha=0.5)
df_2.Целевая_категория.hist(width = 0.2,alpha=0.5)
df_3.Целевая_категория.hist(width = 0.4,alpha=0.5)
df_4.Целевая_категория.hist(width = 0.5,alpha=0.5)
df_5.Целевая_категория.hist(width = 0.6,alpha=0.5)
```



6. Построение графика средних долей правильных значений в зависимости от способа удаления пропусков

5 датафреймов:

`df_1` - Исходные данные без удаления пропусков

`df_2` - Удаление пропусков первым методом (метод заполнения)

`df_3` - Удаление строк с пропущенными данными (функция dropna)

`df_4` - Выполнение `df_3` и последующее удаление дубликатов

`df_5` - Выполнение `df_4` и последующее удаление противоречивых данных

```
        # Создание списка из 5-ти датасетов и пустого списка для результатов обучения

lst_2 = [df_1, df_2, df_3, df_4, df_5]
res_test = []


def to_array(x):
    array = x.values
    X = array[:,0:9]
    Y = array[:,9]
    return [X,Y]
```

```
model = RandomForestClassifier()


# Создание списка с результатами обучения модели для создания графика

for i in lst_2:
    sign = to_array(i)[0]
    meaning = to_array(i)[1]
    X_train, X_test, Y_train, Y_test = train_test_split(sign, meaning,
                                    test_size=0.25, shuffle=True, random_state=7)
    model.fit(X_train,Y_train)
    result=model.score(X_test,Y_test)
    res_test.append(result)
res_test
```

```
[0.4495217853347503,
 0.49727317798710957,
 0.4913017396520696,
 0.4517426273458445,
 0.473463687150838]
```

```
       # Графическое изображение

import matplotlib.pyplot as plt
import seaborn as sns

       x = 1,2,3,4,5

       plt.figure(figsize=(12, 7))

plt.title('Доля правильных ответов после разных степеней очистки данных', alpha
=0.5, color='g', fontsize=20, fontstyle='italic',
fontweight='extra bold', linespacing=10)

line1, = plt.plot(x, res_test, 'o-b',label='Исходный данные(тестовая выборка)')
plt.text(0.5, 0.5, 'text')
plt.grid()
plt.legend(fontsize=15)
plt.show()
```

==========

7. RANDOMFORESTCLASSIFIER() + SELECTKBEST

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

lst_1 = list(range(1,10))
lst_2 = list(range(2,9))
k = []

model=RandomForestClassifier()
for i in lst_2:
   kfold=KFold(n_splits=i,random_state=7,shuffle=True)
   for j in lst_1:
      test=SelectKBest(score_func=f_classif, k=j)
      fit=test.fit(X,Y)
      features=fit.transform(X)
      results=cross_val_score(model, features, Y, cv=kfold)
      res = results.mean()
      k.append(res)
      print(f'n_splits: {i} KBest: {j} {res}')
   print(f'i {i} max: {max(k)}')
```

```
      n_splits: 2 KBest: 1 0.4006002491005403
n_splits: 2 KBest: 2 0.4456105124317954
n_splits: 2 KBest: 3 0.4618149690119072
n_splits: 2 KBest: 4 0.46301612454162583
n_splits: 2 KBest: 5 0.43480862191861447
n_splits: 2 KBest: 6 0.4579142187928277
n_splits: 2 KBest: 7 0.4666162594650822
```

```
n_splits: 2 KBest: 8 0.4802711363573332
n_splits: 2 KBest: 9 0.4934740322771773
i 2 max: 0.4934740322771773
n_splits: 3 KBest: 1 0.40060151895423673
n_splits: 3 KBest: 2 0.4465107699423702
n_splits: 3 KBest: 3 0.4556630359118758
n_splits: 3 KBest: 4 0.4688648963950875
n_splits: 3 KBest: 5 0.4432102372776674
n_splits: 3 KBest: 6 0.4738160668835907
n_splits: 3 KBest: 7 0.4694664423668842
n_splits: 3 KBest: 8 0.47891948132228807
n_splits: 3 KBest: 9 0.4868716272797924
i 3 max: 0.4934740322771773
n_splits: 4 KBest: 1 0.4006041828849116
n_splits: 4 KBest: 2 0.4504137587848577
n_splits: 4 KBest: 3 0.4571651095951278
n_splits: 4 KBest: 4 0.46961802117367646
n_splits: 4 KBest: 5 0.4418606074703426
n_splits: 4 KBest: 6 0.46841736094557807
n_splits: 4 KBest: 7 0.4664677508675936
n_splits: 4 KBest: 8 0.4751697019539669
n_splits: 4 KBest: 9 0.4907732439106416
i 4 max: 0.4934740322771773
n_splits: 5 KBest: 1 0.4006001500375094
n_splits: 5 KBest: 2 0.44651162790697674
n_splits: 5 KBest: 3 0.4580645161290322
n_splits: 5 KBest: 4 0.46526631657914475
n_splits: 5 KBest: 5 0.44441110277569396
n_splits: 5 KBest: 6 0.47216804201050266
n_splits: 5 KBest: 7 0.4705176294073519
n_splits: 5 KBest: 8 0.4841710427606902
n_splits: 5 KBest: 9 0.4928732183045762
i 5 max: 0.4934740322771773
n_splits: 6 KBest: 1 0.40059951941140065
n_splits: 6 KBest: 2 0.44275981652219265
n_splits: 6 KBest: 3 0.4535610317788535
n_splits: 6 KBest: 4 0.46961385327721966
n_splits: 6 KBest: 5 0.4418578344320918
n_splits: 6 KBest: 6 0.4792173811975793
n_splits: 6 KBest: 7 0.46556371853401557
n_splits: 6 KBest: 8 0.48896849144373894
n_splits: 6 KBest: 9 0.4925720950473426
i 6 max: 0.4934740322771773
n_splits: 7 KBest: 1 0.4006059401725013
n_splits: 7 KBest: 2 0.4409634262939173
n_splits: 7 KBest: 3 0.4532667736559472
n_splits: 7 KBest: 4 0.4670693041119911
n_splits: 7 KBest: 5 0.4418598499210807
n_splits: 7 KBest: 6 0.4798209420179405
n_splits: 7 KBest: 7 0.4729208577449868
n_splits: 7 KBest: 8 0.4816186705532161
n_splits: 7 KBest: 9 0.49422339764867124
i 7 max: 0.49422339764867124
n_splits: 8 KBest: 1 0.40060592582356686
n_splits: 8 KBest: 2 0.4397647548227924
```

```
n_splits: 8 KBest: 3 0.4549163190456038
n_splits: 8 KBest: 4 0.46841974199751846
n_splits: 8 KBest: 5 0.4487606193556559
n_splits: 8 KBest: 6 0.47936836605145655
n_splits: 8 KBest: 7 0.4744205236051255
n_splits: 8 KBest: 8 0.4867198678032364
n_splits: 8 KBest: 9 0.49017268778014805
i 8 max: 0.49422339764867124
```

Вывод: отбор показал, что лучший результат дает выбор 7 лучших признаков

```
test=SelectKBest(score_func=f_classif, k=6)
fit=test.fit(X,Y)
features=fit.transform(X)
fit.scores_
```

```
array([  4.70368846, 496.43200214, 418.45793796, 354.51376472,
        53.09011086,  20.75182914, 364.22358787,  92.0107774 ,
        33.2485345 ])
```

```
features[0:2]
```

```
array([[ 0., 22.,  0.,  2.,  3.,  4.],
       [ 1., 67.,  1.,  4.,  3.,  1.]])
```

Вывод: лучшими 7-ю признаками будут: **'Семейное_положение', 'Возраст', 'Высшее_образование', 'Профессия', 'Уровень_расходов', 'Размер_семьи', 'Анонимная_категория'**

8. RANDOMFORESTCLASSIFIER() + РЕКУРСИВНОЕ_ИСКЛЮЧЕНИЕ

```
from sklearn.feature_selection import RFE
```

```
lst_1 = list(range(1,10))
lst_2 = list(range(2,9))
k = []
```

```
model=model=RandomForestClassifier()
for j in lst_1:
    rfe=RFE(model,n_features_to_select=j)
    fit=rfe.fit(X,Y)
    features=fit.transform(X)
    results=cross_val_score(model, features, Y, cv=7)
    res = results.mean()
    k.append(res)
    print(f'n_splits: {i} KBest: {j} {res}')
print(f'i {i} max: {max(k)}')
```

```
n_splits: 4 KBest: 1 0.43420741853929407
n_splits: 4 KBest: 2 0.4655635076695946
n_splits: 4 KBest: 3 0.41800754929463907
n_splits: 4 KBest: 4 0.4393115063444056
n_splits: 4 KBest: 5 0.4427580056156776
n_splits: 4 KBest: 6 0.47081781295939157
```

```
n_splits: 4 KBest: 7 0.4792184974724413
n_splits: 4 KBest: 8 0.48927236162040894
n_splits: 4 KBest: 9 0.49437314275132926
i 4 max: 0.49437314275132926
```

Вывод: отбор показал, что лучший результат дает выбор 9 лучших признаков

```
model=LogisticRegression(solver='newton-cg',class_weight= 'balanced')
rfe=RFE(model,n_features_to_select=7)
fit=rfe.fit(X,Y)
fit.n_features_

fit.support_

array([ True,  True, False,  True,  True, False,  True,  True,  True])

fit.ranking_

array([1, 1, 3, 1, 1, 2, 1, 1, 1])
```

9. RANDOMFORESTCLASSIFIER() + ОТБОР_ГЛАВНЫХ_КОМПОНЕНТ

```
from sklearn.decomposition import PCA

lst_1 = list(range(1,10))
lst_2 = list(range(2,9))
k = []

model=model=model=RandomForestClassifier()
for j in lst_1:
    pca=PCA(n_components=j)
    fit=pca.fit(X)
    features=pca.fit_transform(X)
    results=cross_val_score(model, features, Y, cv=7)
    res = results.mean()
    k.append(res)
    print(f'n_splits: {i} KBest: {j} {res}')
print(f'i {i} max: {max(k)}')

n_splits: 4 KBest: 1 0.3629405592247393
n_splits: 4 KBest: 2 0.3909959576695316
n_splits: 4 KBest: 3 0.44261361417599565
n_splits: 4 KBest: 4 0.4537141194358121
n_splits: 4 KBest: 5 0.47216709349007185
n_splits: 4 KBest: 6 0.4894199022736062
n_splits: 4 KBest: 7 0.49752471817688254
n_splits: 4 KBest: 8 0.49647351070543644
n_splits: 4 KBest: 9 0.48837200147635135
i 4 max: 0.49752471817688254
```

Вывод: отбор показал, что лучший результат дает выбор 9 лучших признаков

10. Метод k-средних, результат кластеризации и создание графика пересечения кластеров и результативных признаков

```
        # Импортируем библиотеки
from sklearn import datasets
from sklearn.cluster import KMeans

        # Описываем модель
model = KMeans(n_clusters=2)

        # Проводим моделирование
result = model.fit(X)

        # Предсказание на единичном примере
predicted_label = model.predict([[1,0.0,22,0.0,2.0,1.0,3,4.0,4.0]])
predicted_label

        array([0])

        # Предсказание на всем наборе данных
all_predictions = model.predict(X)
#all_predictions.unique()

        # Выводим предсказания
print(predicted_label)
print(all_predictions)

[0]
[1 0 0 ... 0 0 0]

        # KA_knn - это столбец с результатами кластерного анализа

df_KA_knn[df_KA_knn['Целевая_категория'] == 4].KA_knn.value_counts()

        2    1525
0     412
1     180
3     151
Name: KA_knn, dtype: int64

df_KA_knn[df_KA_knn['Целевая_категория'] == 3].KA_knn.value_counts()

        3    795
0    580
1    302
2    293
Name: KA_knn, dtype: int64

df_KA_knn[df_KA_knn['Целевая_категория'] == 2].KA_knn.value_counts()

        0    683
3    608
1    293
2    274
Name: KA_knn, dtype: int64
```

```
df_KA_knn[df_KA_knn['Целевая_категория'] == 1].KA_knn.value_counts()
```

```
       0    780
2    503
3    384
1    304
Name: KA_knn, dtype: int64
```

ГРАФИЧЕСКА ЧАСТЬ:

```
import matplotlib.pyplot as plt
import seaborn as sns

fg = plt.figure(figsize=(12,12),constrained_layout=True) # задаем рамку
gs = fg.add_gridspec(2,2) # задаем квадраты в рамке
q = fg.add_subplot(gs[0,0]) # задаем рамку для таблицы

data = df_KA_knn[df_KA_knn['Целевая_категория'] == 4].KA_knn.value_counts()
labels = ['кластер_1', 'кластер_2', 'кластер_3', 'кластер_4']

# Настройка цветовой гаммы
colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

# Рендеринг круговой диаграммы
g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
        labels=labels, pctdistance=0.5, shadow =True, startangle = 290
       )
# Название рисунка
plt.figtext(0.05, -0.1, 'Рисунок 1')

q = fg.add_subplot(gs[0,1]) # задаем рамку для таблицы

data = df_KA_knn[df_KA_knn['Целевая_категория'] == 3].KA_knn.value_counts()
labels = ['кластер_1', 'кластер_2', 'кластер_3', 'кластер_4']

# Настройка цветовой гаммы
colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

# Рендеринг круговой диаграммы
g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
        labels=labels, pctdistance=0.5, shadow =True, startangle = 290
       )

q = fg.add_subplot(gs[1,0]) # задаем рамку для таблицы

data = df_KA_knn[df_KA_knn['Целевая_категория'] == 2].KA_knn.value_counts()
labels = ['кластер_1', 'кластер_2', 'кластер_3', 'кластер_4']

# Настройка цветовой гаммы
colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

# Рендеринг круговой диаграммы
g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
        labels=labels, pctdistance=0.5, shadow =True, startangle = 290
       )
```

```
q = fg.add_subplot(gs[1,1]) # задаем рамку для таблицы

data = df_KA_knn[df_KA_knn['Целевая_категория'] == 1].KA_knn.value_counts()
labels = ['кластер_1', 'кластер_2', 'кластер_3', 'кластер_4']

# Настройка цветовой гаммы
colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

# Рендеринг круговой диаграммы
g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
        labels=labels, pctdistance=0.5, shadow =True, startangle = 290
       )

# Название рисунка
```

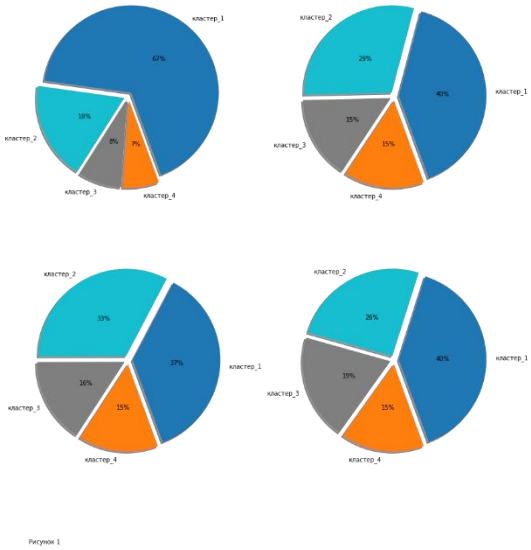


Рисунок 1

11. Комбинирование метода SelectKBest и метод k-средних для перебора признаков от 1 до 9, проведения кластерного анализа на их основе и создание графиков пересечения кластеров и результативных признаков в зависимости от числа признаков.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

lst_1 = list(range(2,10))
lst_2 = list(range(2,9))
lst_3 = list(range(1,5))
k = []
```

```
df_KA_knn = df
for i in lst_1:
# выбираем лучших k-признаков
    test=SelectKBest(score_func=f_classif, k=i)
    fit=test.fit(X,Y)
    features=fit.transform(X)
# кластерный анализ, метод KNN
    model = KMeans(n_clusters=4)
    result = model.fit(features)
    all_predictions = model.predict(features)
    df_KA_knn['KA_knn'] = pd.Series(all_predictions, index=df_KA_knn.index)
    fg = plt.figure(figsize=(12,12),constrained_layout=True) # задаем рамку
    gs = fg.add_gridspec(2,2) # задаем квадраты в рамке
    q = fg.add_subplot(gs[0,0]) # задаем рамку для таблицы

    data = df_KA_knn[df_KA_knn['Целевая_категория']==4]['KA_knn'].value_counts(
)
    labels = ['кластер_1', 'кластер_2', 'кластер_3', 'кластер_4']

    # Настройка цветовой гаммы
    colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

    # Рендеринг круговой диаграммы
    g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
            labels=labels, pctdistance=0.5, shadow =True, startangle = 290
           )
    # Название рисунка
    plt.figtext(0.05, -0.1, 'Рисунок 1')

    q = fg.add_subplot(gs[0,1]) # задаем рамку для таблицы

    data = df_KA_knn[df_KA_knn['Целевая_категория'] == 3]['KA_knn'].value_count
s()
    labels = ['кластер_1', 'кластер_2', 'кластер_3', 'кластер_4']

    # Настройка цветовой гаммы
    colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

    # Рендеринг круговой диаграммы
    g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
            labels=labels, pctdistance=0.5, shadow =True, startangle = 290
           )

        q = fg.add_subplot(gs[1,0]) # задаем рамку для таблицы

    data = df_KA_knn[df_KA_knn['Целевая_категория'] == 2]['KA_knn'].value_count
s()
    labels = ['кластер_1', 'кластер_2', 'кластер_3', 'кластер_4']

    # Настройка цветовой гаммы
    colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

    # Рендеринг круговой диаграммы
    g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
            labels=labels, pctdistance=0.5,shadow =True, startangle = 290)
```

```
    q = fg.add_subplot(gs[1,1]) # задаем рамку для таблицы

    data = df_KA_knn[df_KA_knn['Целевая_категория'] == 1]['KA_knn'].value_count
s()
    labels = ['кластер_1', 'кластер_2', 'кластер_3', 'кластер_4']

    # Настройка цветовой гаммы
    colors = ['tab:blue', 'tab:cyan', 'tab:gray', 'tab:orange', 'tab:red']

    # Рендеринг круговой диаграммы
    g = plt.pie(data, colors = colors, autopct='%.0f%%',explode=[0.05]*4,
            labels=labels, pctdistance=0.5, shadow =True, startangle = 290
           )
```

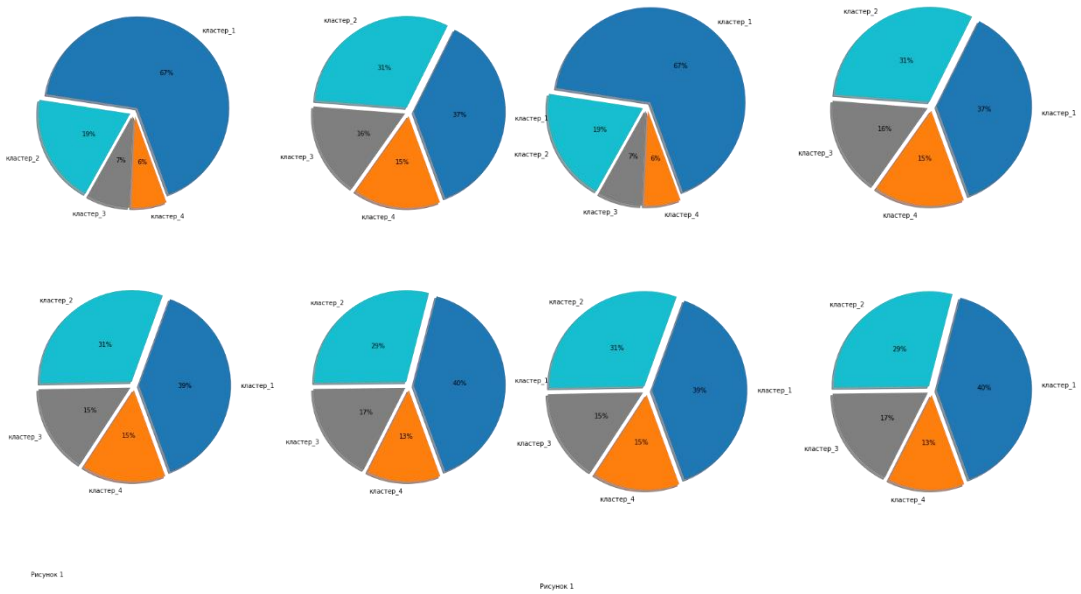


Рисунок 1

Рисунок 1

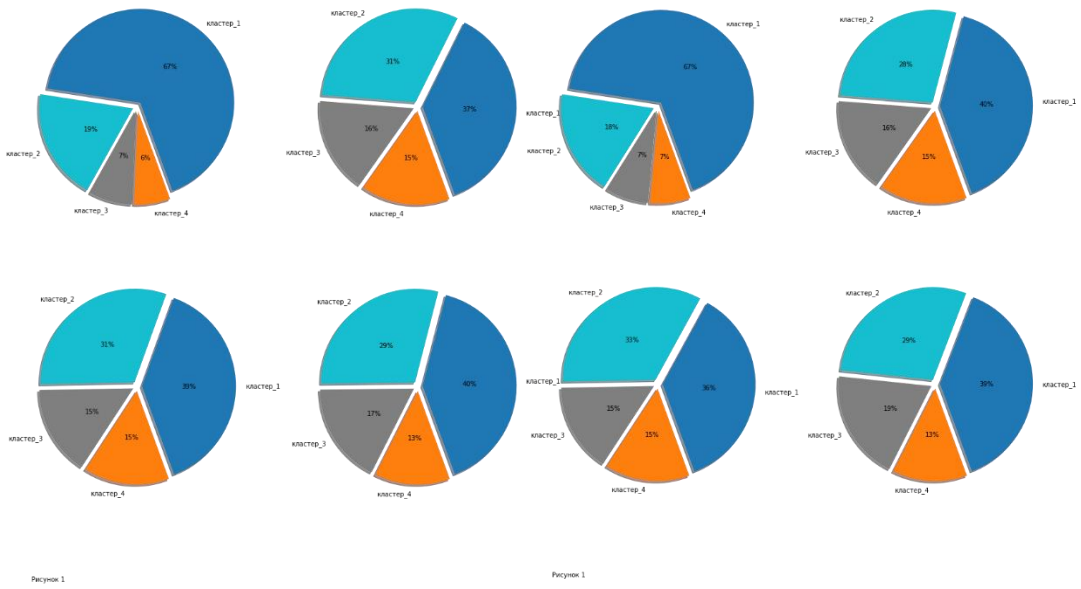
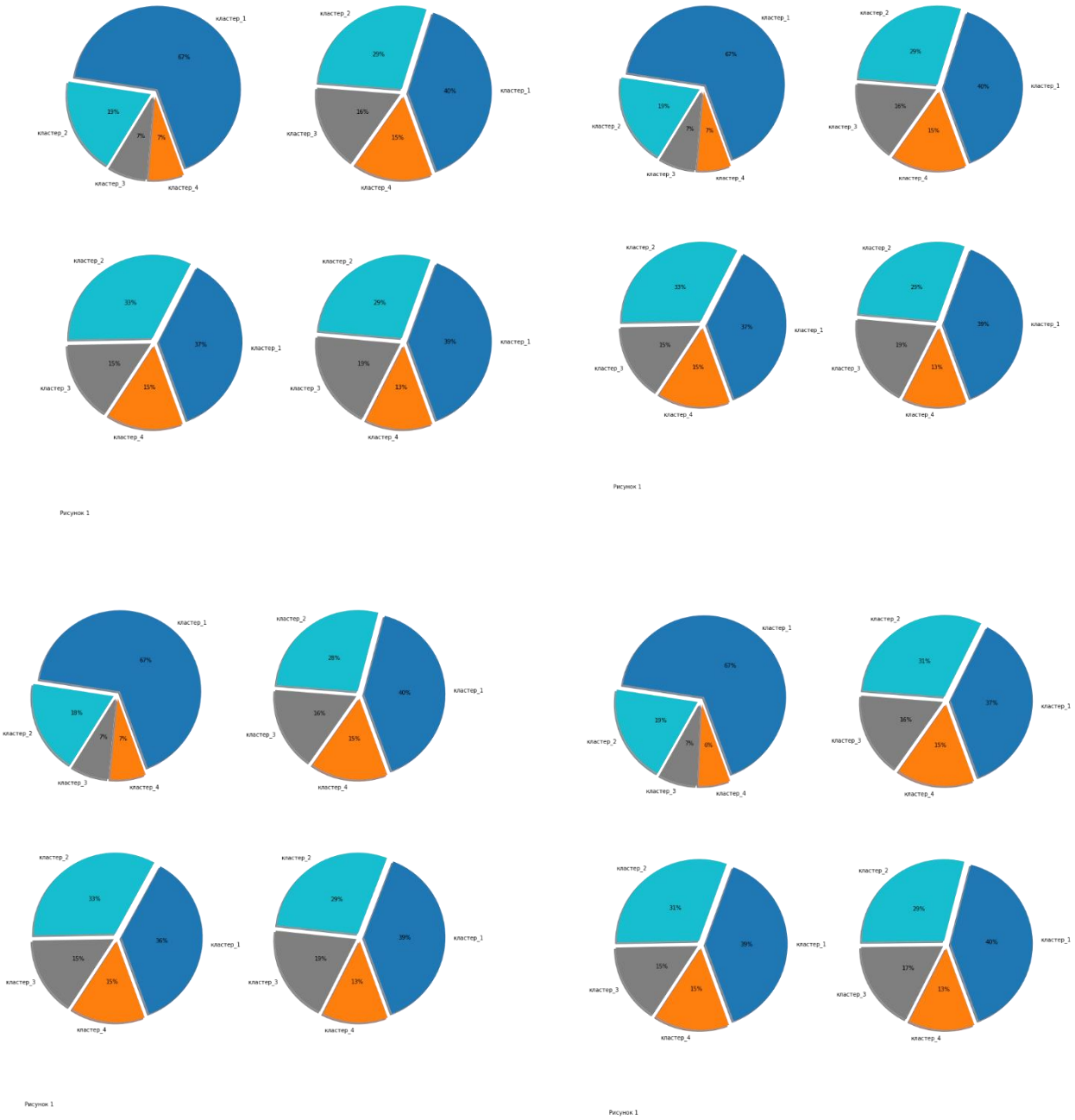


Рисунок 1

Рисунок 1



12. Иерархический кластерный анализ

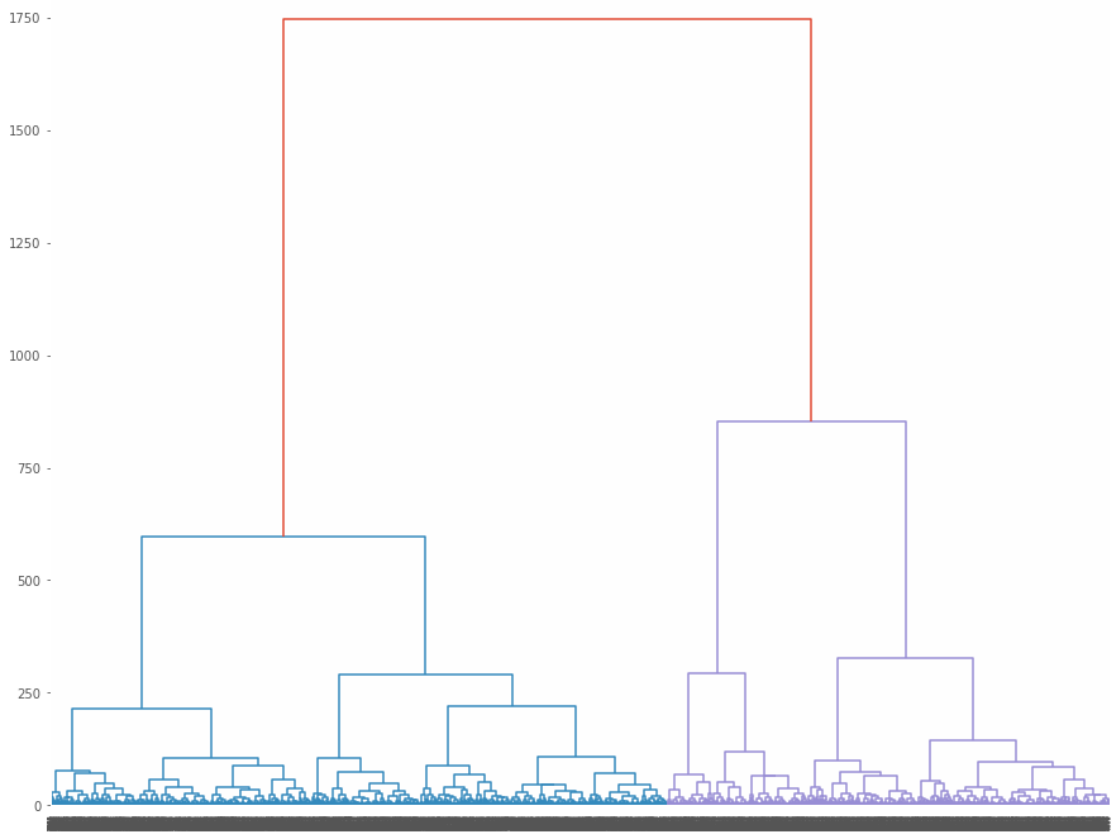
```
        import os
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn import preprocessing
from sklearn.cluster import KMeans
import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')
```

```
link = linkage(df.iloc[:,:9], 'ward', 'euclidean')

link[:5]

array([[6.045e+03, 6.145e+03, 0.000e+00, 2.000e+00],
       [5.975e+03, 6.031e+03, 0.000e+00, 2.000e+00],
       [5.956e+03, 6.007e+03, 0.000e+00, 2.000e+00],
       [5.932e+03, 5.953e+03, 0.000e+00, 2.000e+00],
       [5.958e+03, 8.070e+03, 0.000e+00, 3.000e+00]])

plt.figure(figsize=(15, 12))
dn = dendrogram(link)
```



13. Поиск гиперпараметров с помощью GrideSearchCV на примере модели BaggingClassifier

```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, shuff
le=True, random_state=7)

clf = BaggingClassifier()

params = {
          'bootstrap': [True, False],
          'bootstrap_features': [True, False],
          'n_estimators': [5, 10, 15,100,1000],
          'max_samples' : [0.6, 0.8, 1.0],
          'base_estimator' : [DecisionTreeClassifier(criterion='entropy', max_d
epth=5),
                              KNeighborsClassifier(metric='manhattan', n_neighb
ors=50)]
}

grid = GridSearchCV(clf, param_grid = params, cv=3,verbose=1,n_jobs=-1)

grid.fit(X_train,Y_train)

Fitting 3 folds for each of 120 candidates, totalling 360 fits


GridSearchCV(cv=3, estimator=BaggingClassifier(), n_jobs=-1,
             param_grid={'base_estimator':

[DecisionTreeClassifier(criterion='entropy', max_depth=5),
KNeighborsClassifier(metric='manhattan', n_neighbors=50)],
                         'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_samples': [0.6, 0.8, 1.0],
                         'n_estimators': [5, 10, 15, 100, 1000]},
             verbose=1)

print(grid.best_params_)
print(grid.best_estimator_)

BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entropy',
                                                        max_depth=5),
                  bootstrap=False, bootstrap_features=True, n_estimators=100)
```

```
from sklearn.metrics import classification_report, confusion_matrix

grid_predictions = grid.predict(X_test)
print(classification_report(Y_test, grid_predictions))
```

```
                precision    recall  f1-score   support

         1.0       0.46      0.48      0.47       351
         2.0       0.42      0.28      0.34       347
         3.0       0.54      0.57      0.55       361
         4.0       0.59      0.71      0.64       373

    accuracy                           0.51      1432
   macro avg       0.50      0.51      0.50      1432
weighted avg       0.50      0.51      0.50      1432
```

```
print(confusion_matrix(Y_test, grid_predictions))
```

```
[[169  56  48  78]
 [ 86  97 118  46]
 [ 37  56 205  63]
 [ 77  22  10 264]]
```

```
       kfold=KFold(n_splits=3,random_state=7,shuffle=True)
results=cross_val_score(model, X, Y, cv=3)
results.mean()
```

```
       0.5105639951108784
```

```
       ================================================================================
========================================
```

14. Комбинирование метода SelectKBest и моделей классификации из списка для перебора признаков от 1 до 9 для каждой из моделей. Сначала обучались модели с параметрами по умолчанию, а затем модели с подобранными параметрами.

```
       from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC

lst_1 = list(range(1,10))
lst_3 = [LogisticRegression(max_iter = 5000), DecisionTreeClassifier(), AdaBoos
tClassifier(),SVC(),
         KNeighborsClassifier(),RandomForestClassifier(),ExtraTreesClassifier()
,
         XGBClassifier(seed = 7, n_estimators = 100, max_depth = 6, learning_ra
te = 0.3),
        GradientBoostingClassifier(n_estimators=20, random_state=7),
        BaggingClassifier()]
k = []

for i in lst_3:
   model = i
   for j in lst_1:
        test=SelectKBest(score_func=f_classif, k=j)
        fit=test.fit(X,Y)
        features=fit.transform(X)
        X_train, X_test, Y_train, Y_test = train_test_split(features, Y, test_s
ize=0.25, shuffle=True, random_state=7)
        model.fit(X_train,Y_train)
        result=model.score(X_test,Y_test)
        k.append(result)
        print(result)
        print(f'Модель: {i} Количество  признаков = {j}')
```

```
0.3737252549490102
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 1
0.3869226154769046
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 2
0.4295140971805639
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 3
0.42651469706058787
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 4
0.4475104979004199
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 5
0.4595080983803239
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 6
0.46550689862027594
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 7
0.46550689862027594
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 8
0.4631073785242951
Модель: LogisticRegression(max_iter=5000) Количество  признаков = 9
```

```
0.3737252549490102
Модель: DecisionTreeClassifier() Количество  признаков = 1
0.4403119376124775
Модель: DecisionTreeClassifier() Количество  признаков = 2
0.4505098980203959
Модель: DecisionTreeClassifier() Количество  признаков = 3
0.45410917816436713
Модель: DecisionTreeClassifier() Количество  признаков = 4
0.4319136172765447
Модель: DecisionTreeClassifier() Количество  признаков = 5
0.44631073785242953
Модель: DecisionTreeClassifier() Количество  признаков = 6
0.4325134973005399
Модель: DecisionTreeClassifier() Количество  признаков = 7
0.4403119376124775
Модель: DecisionTreeClassifier() Количество  признаков = 8
0.4319136172765447
Модель: DecisionTreeClassifier() Количество  признаков = 9
0.3737252549490102
Модель: AdaBoostClassifier() Количество  признаков = 1
0.4391121775644871
Модель: AdaBoostClassifier() Количество  признаков = 2
0.4559088182363527
Модель: AdaBoostClassifier() Количество  признаков = 3
0.4793041391721656
Модель: AdaBoostClassifier() Количество  признаков = 4
0.47690461907618475
Модель: AdaBoostClassifier() Количество  признаков = 5
0.5020995800839833
Модель: AdaBoostClassifier() Количество  признаков = 6
0.4979004199160168
Модель: AdaBoostClassifier() Количество  признаков = 7
0.5050989802039592
Модель: AdaBoostClassifier() Количество  признаков = 8
0.5032993401319736
Модель: AdaBoostClassifier() Количество  признаков = 9
0.3737252549490102
Модель: SVC() Количество  признаков = 1
0.4259148170365927
Модель: SVC() Количество  признаков = 2
0.4313137372525495
Модель: SVC() Количество  признаков = 3
0.4295140971805639
Модель: SVC() Количество  признаков = 4
0.4325134973005399
Модель: SVC() Количество  признаков = 5
0.4439112177564487
Модель: SVC() Количество  признаков = 6
0.4475104979004199
Модель: SVC() Количество  признаков = 7
0.44571085782843434
Модель: SVC() Количество  признаков = 8
0.4475104979004199
Модель: SVC() Количество  признаков = 9
0.36532693461307736
```

```
Модель: KNeighborsClassifier() Количество  признаков = 1
0.385122975404919
Модель: KNeighborsClassifier() Количество  признаков = 2
0.4295140971805639
Модель: KNeighborsClassifier() Количество  признаков = 3
0.4451109778044391
Модель: KNeighborsClassifier() Количество  признаков = 4
0.4361127774445111
Модель: KNeighborsClassifier() Количество  признаков = 5
0.46370725854829037
Модель: KNeighborsClassifier() Количество  признаков = 6
0.4391121775644871
Модель: KNeighborsClassifier() Количество  признаков = 7
0.4487102579484103
Модель: KNeighborsClassifier() Количество  признаков = 8
0.4409118176364727
Модель: KNeighborsClassifier() Количество  признаков = 9
0.3737252549490102
Модель: RandomForestClassifier() Количество  признаков = 1
0.44211157768446313
Модель: RandomForestClassifier() Количество  признаков = 2
0.4505098980203959
Модель: RandomForestClassifier() Количество  признаков = 3
0.46130773845230955
Модель: RandomForestClassifier() Количество  признаков = 4
0.4367126574685063
Модель: RandomForestClassifier() Количество  признаков = 5
0.46250749850029993
Модель: RandomForestClassifier() Количество  признаков = 6
0.45830833833233353
Модель: RandomForestClassifier() Количество  признаков = 7
0.4631073785242951
Модель: RandomForestClassifier() Количество  признаков = 8
0.489502099580084
Модель: RandomForestClassifier() Количество  признаков = 9
0.3737252549490102
Модель: ExtraTreesClassifier() Количество  признаков = 1
0.44151169766046 79
Модель: ExtraTreesClassifier() Количество  признаков = 2
0.4505098980203959
Модель: ExtraTreesClassifier() Количество  признаков = 3
0.45350929814037194
Модель: ExtraTreesClassifier() Количество  признаков = 4
0.43431313737252547
Модель: ExtraTreesClassifier() Количество  признаков = 5
0.4631073785242951
Модель: ExtraTreesClassifier() Количество  признаков = 6
0.4589082183563287
Модель: ExtraTreesClassifier() Количество  признаков = 7
0.46610677864427114
Модель: ExtraTreesClassifier() Количество  признаков = 8
0.4709058188362327
Модель: ExtraTreesClassifier() Количество  признаков = 9
0.3737252549490102
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
```

```
признаков = 1
0.4355128974205159
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
признаков = 2
0.4553089382123575
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
признаков = 3
0.48890221955608876
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
признаков = 4
0.4787042591481704
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
признаков = 5
0.5230953809238152
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
признаков = 6
0.5164967006598681
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
признаков = 7
0.5218956208758249
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
признаков = 8
0.5206958608278345
Модель: GradientBoostingClassifier(n_estimators=20, random_state=7) Количество
признаков = 9
0.3737252549490102
Модель: BaggingClassifier() Количество  признаков = 1
0.4331133773245351
Модель: BaggingClassifier() Количество  признаков = 2
0.4469106178764247
Модель: BaggingClassifier() Количество  признаков = 3
0.4595080983803239
Модель: BaggingClassifier() Количество  признаков = 4
0.4337132573485303
Модель: BaggingClassifier() Количество  признаков = 5
0.44991001799640074
Модель: BaggingClassifier() Количество  признаков = 6
0.4553089382123575
Модель: BaggingClassifier() Количество  признаков = 7
0.46250749850029993
Модель: BaggingClassifier() Количество  признаков = 8
0.4673065386922616
Модель: BaggingClassifier() Количество  признаков = 9
```

```
k = np.around(k,4)
print(len(k))
k
```

```
array([0.3737, 0.3869, 0.4295, 0.4265, 0.4475, 0.4595, 0.4655, 0.4655,
       0.4631, 0.3737, 0.4403, 0.4505, 0.4541, 0.4319, 0.4463, 0.4325,
       0.4403, 0.4319, 0.3737, 0.4391, 0.4559, 0.4793, 0.4769, 0.5021,
       0.4979, 0.5051, 0.5033, 0.3737, 0.4259, 0.4313, 0.4295, 0.4325,
       0.4439, 0.4475, 0.4457, 0.4475, 0.3653, 0.3851, 0.4295, 0.4451,
       0.4361, 0.4637, 0.4391, 0.4487, 0.4409, 0.3737, 0.4421, 0.4505,
       0.4613, 0.4367, 0.4625, 0.4583, 0.4631, 0.4895, 0.3737, 0.4415,
       0.4505, 0.4535, 0.4343, 0.4631, 0.4589, 0.4661, 0.4709, 0.3737,
```

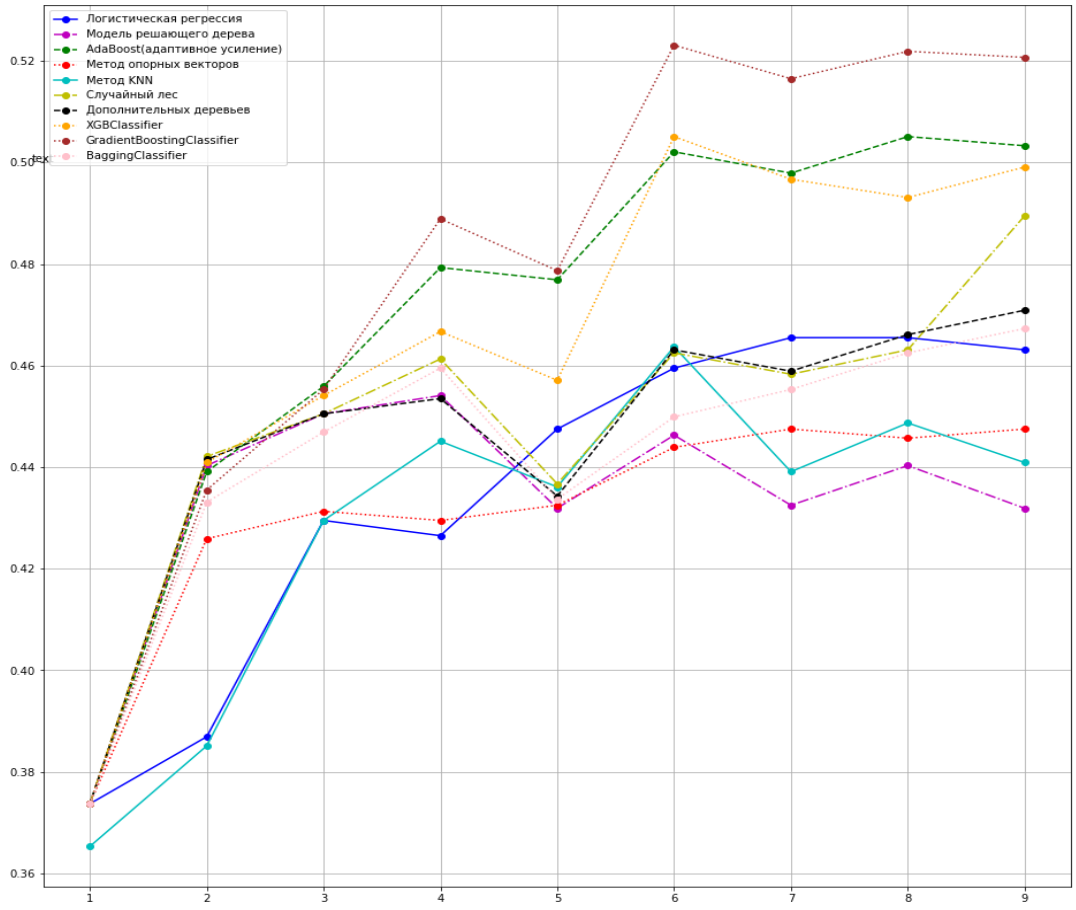
```
        0.4409, 0.4541, 0.4667, 0.4571, 0.5051, 0.4967, 0.4931, 0.4991,
        0.3737, 0.4355, 0.4553, 0.4889, 0.4787, 0.5231, 0.5165, 0.5219,
        0.5207, 0.3737, 0.4331, 0.4469, 0.4595, 0.4337, 0.4499, 0.4553,
        0.4625, 0.4673])

import matplotlib.pyplot as plt
import seaborn as sns
x = 1,2,3,4,5,6,7,8,9
plt.figure(figsize=(16, 16))

line1, = plt.plot(x, k[:9], 'o-b',label='Логистическая регрессия')
line2, = plt.plot(x, k[9:18], 'o-.m', label='Модель решающего дерева')
line3, = plt.plot(x, k[18:27], 'o--g', label='AdaBoost(адаптивное усиление)')

line4, = plt.plot(x, k[27:36], 'o:r', label='Метод опорных векторов')
line5, = plt.plot(x, k[36:45], 'o-c', label='Метод KNN')
line6, = plt.plot(x, k[45:54], 'o-.y', label='Случайный лес')
line7, = plt.plot(x, k[54:63], 'o--k', label='Дополнительных деревьев')
line8, = plt.plot(x, k[63:72], 'o:',color='orange', label='XGBClassifier')
line9, = plt.plot(x, k[72:81], 'o:',color='brown', label='GradientBoostingClass
ifier')
line10, = plt.plot(x, k[81:], 'o:',color='pink', label='BaggingClassifier')

plt.text(0.5, 0.5, 'text')
plt.grid()
plt.legend(fontsize=10)
plt.show()
```



```
lst_1 = list(range(1,10))
lst_3 = [LogisticRegression(C=0.01, max_iter=5000),
         DecisionTreeClassifier(criterion='entropy', max_depth=5),
         AdaBoostClassifier(learning_rate=0.11, n_estimators=60),
         SVC(C=100, gamma=0.001),
         KNeighborsClassifier(metric='manhattan', n_neighbors=50),
         RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=6
0),
         ExtraTreesClassifier(max_depth=37,
                              min_weight_fraction_leaf=0.001,
                              n_estimators=52),
         XGBClassifier(learning_rate=0.2, max_depth=5, max_features='log2',
                           min_samples_leaf=0.1, min_samples_split=0.1,
                           n_estimators=10),
         GradientBoostingClassifier(n_estimators=100, random_state=7),
         BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='ent
ropy',
                                                                 max_depth=5),
                           bootstrap=False, bootstrap_features=True, max_samples=0.8,
                           n_estimators=100)]
k = []

for i in lst_3:
   model = i
   for j in lst_1:
        test=SelectKBest(score_func=f_classif, k=j)
        fit=test.fit(X,Y)
        features=fit.transform(X)
        X_train, X_test, Y_train, Y_test = train_test_split(features, Y, test_s
ize=0.25,
                                                            shuffle=True, rando
m_state=7)
        model.fit(X_train,Y_train)
        result=model.score(X_test,Y_test)
        k.append(result)
        print(result)
        print(f'Модель: {i} Количество  признаков = {j}')
```

```
0.40109072880515617
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 1
0.4258800198314328
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 2
0.44719881011403073
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 3
0.4516608824987605
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 4
0.47000495785820523
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 5
0.47992067426871593
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 6
```

```
0.47892910262766486
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 7
0.47992067426871593
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 8
0.47645017352503716
Модель: LogisticRegression(C=0.01, max_iter=5000) Количество  признаков = 9
0.40109072880515617
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 1
0.4526524541398116
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 2
0.47645017352503716
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 3
0.4992563212692117
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 4
0.503222607833416
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 5
0.5270203272186416
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 6
0.5230540406544373
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 7
0.5230540406544373
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 8
0.5230540406544373
Модель: DecisionTreeClassifier(criterion='entropy', max_depth=5) Количество  пр
изнаков = 9
0.40109072880515617
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 1
0.4417451660882499
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 2
0.4645513138324244
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 3
0.4784333168071393
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 4
0.4784333168071393
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 5
0.5027268220128904
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 6
0.503222607833416
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 7
0.503222607833416
```

```
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 8
0.503222607833416
Модель: AdaBoostClassifier(learning_rate=0.11, n_estimators=60) Количество  при
знаков = 9
0.40109072880515617
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 1
0.4452156668319286
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 2
0.4630639563708478
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 3
0.473475458601884
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 4
0.47942488844819037
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 5
0.4992563212692117
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 6
0.5022310361923649
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 7
0.5061973227565691
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 8
0.5076846802181457
Модель: SVC(C=100, gamma=0.001) Количество  признаков = 9
0.40109072880515617
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 1
0.43529995042141795
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 2
0.4516608824987605
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 3
0.47892910262766486
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 4
0.480912245909767
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 5
0.5076846802181457
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 6
0.5086762518591968
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 7
0.4923153197818542
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 8
0.5022310361923649
Модель: KNeighborsClassifier(metric='manhattan', n_neighbors=50) Количество  пр
изнаков = 9
0.40109072880515617
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
) Количество  признаков = 1
0.45017352503718394
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
```

```
) Количество  признаков = 2
0.4749628160634606
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
) Количество  признаков = 3
0.4898363906792266
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
) Количество  признаков = 4
0.49876053544868615
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
) Количество  признаков = 5
0.5319781854238969
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
) Количество  признаков = 6
0.5349529003470501
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
) Количество  признаков = 7
0.541398116013882
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
) Количество  признаков = 8
0.5409023301933564
Модель: RandomForestClassifier(max_depth=7, min_samples_leaf=3, n_estimators=60
) Количество  признаков = 9
0.40109072880515617
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 1
0.45066931085770945
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 2
0.4705007436787308
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 3
0.4923153197818542
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 4
0.4853743182944968
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 5
0.5374318294496777
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 6
0.5433812592959841
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 7
0.540406544372831
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 8
0.5458601883986118
Модель: GradientBoostingClassifier(random_state=7) Количество  признаков = 9
0.40109072880515617
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 1
0.4571145265245414
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 2
0.4779375309866138
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
```

```
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 3
0.48834903321765
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 4
0.4928111056023798
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 5
0.5334655428854734
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 6
0.5349529003470501
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 7
0.5349529003470501
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 8
0.5389191869112543
Модель: BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entr
opy',
                                                    max_depth=5),
                  bootstrap=False, bootstrap_features=True, max_samples=0.8,
                  n_estimators=100) Количество  признаков = 9
```

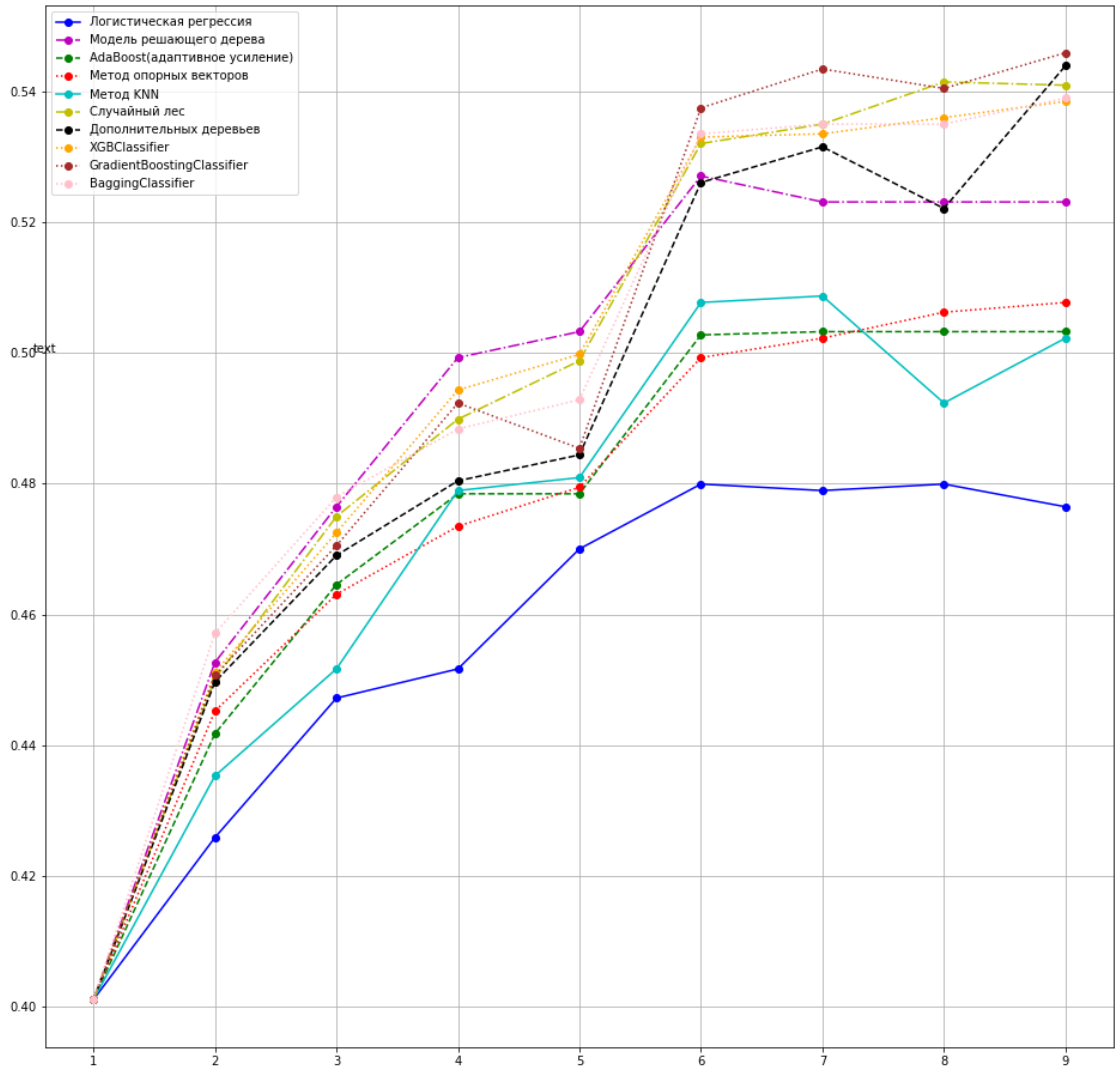
```
import matplotlib.pyplot as plt
import seaborn as sns
x = 1,2,3,4,5,6,7,8,9
plt.figure(figsize=(16, 16))

plt.title('Зависимость качества моделей от числа признаков', alpha=0.5, color='
g', fontsize=20, fontstyle='italic',
fontweight='extra bold', linespacing=10)

line1, = plt.plot(x, k[:9], 'o-b',label='Логистическая регрессия')
line2, = plt.plot(x, k[9:18], 'o-.m', label='Модель решающего дерева')
line3, = plt.plot(x, k[18:27], 'o--g', label='AdaBoost(адаптивное усиление)')
line4, = plt.plot(x, k[27:36], 'o:r', label='Метод опорных векторов')
line5, = plt.plot(x, k[36:45], 'o-c', label='Метод KNN')
```

```
line6, = plt.plot(x, k[45:54], 'o-.y', label='Случайный лес')
line7, = plt.plot(x, k[54:63], 'o--k', label='Дополнительных деревьев')
line8, = plt.plot(x, k[63:72], 'o:',color='orange', label='XGBClassifier')
line9, = plt.plot(x, k[72:81], 'o:',color='brown', label='GradientBoostingClass
ifier')
line10, = plt.plot(x, k[81:], 'o:',color='pink', label='BaggingClassifier')

plt.text(0.5, 0.5, 'text')
plt.grid()
plt.legend(fontsize=10)
plt.show()
```



15. Кластеризация методом k-средних, обучение и оценка модели модели.

```
from sklearn import datasets
from sklearn.cluster import KMeans

model = KMeans(n_clusters=4)
```

```
result = model.fit(X)

all_predictions = model.predict(X)

print(all_predictions)

[0 0 0 ... 2 2 2]

all_predictions

array([0, 0, 0, ..., 2, 2, 2])

df_KA_knn = df

df_KA_knn['KA_knn'] = pd.Series(all_predictions, index=df_KA_knn.index)

array = df_KA_knn.values
array

array([[ 1.,  0., 18., ...,  6.,  1.,  0.],
       [ 1.,  1., 18., ...,  6.,  1.,  0.],
       [ 0.,  0., 19., ...,  4.,  1.,  0.],
       ...,
       [ 0.,  1., 89., ...,  6.,  4.,  2.],
       [ 1.,  1., 89., ...,  6.,  4.,  2.],
       [ 0.,  1., 89., ...,  6.,  4.,  2.]])

X = array[:,0:9]
Y = array[:,10]

Y

array([0., 0., 0., ..., 2., 2., 2.])



from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
# отбор признаков:
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
# ансамблевые функции:
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
import numpy as np
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import GridSearchCV
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
```

```
# сохраняет параметры после перезагрузки:
import pickle
import joblib
from sklearn.metrics import classification_report, confusion_matrix

from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier



X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, shuff
le=True, random_state=7)

model = GradientBoostingClassifier(n_estimators=100, random_state=7)

model.fit(X_train,Y_train)

GradientBoostingClassifier(random_state=7)

new_predictions = model.predict(X_test)

print(classification_report(Y_test, new_predictions))
print(confusion_matrix(Y_test, new_predictions))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.99      1.00       585
         1.0       1.00      1.00      1.00       526
         2.0       1.00      1.00      1.00       255
         3.0       1.00      1.00      1.00       651

    accuracy                           1.00      2017
   macro avg       1.00      1.00      1.00      2017
weighted avg       1.00      1.00      1.00      2017

[[582   0   0   3]
 [  0 526   0   0]
 [  0   0 255   0]
 [  0   2   0 649]]
```

```
       kfold=KFold(n_splits=3,random_state=7,shuffle=True)
results=cross_val_score(model, X, Y, cv=3)
results.mean()
```

```
       0.9988843436221644
```